Qualitative and Quantitative Security Analyses for ZigBee Wireless Sensor Networks

Yuksel, Ender

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

Qualitative and Quantitative Security Analyses for ZigBee Wireless Sensor Networks

Ender Yüksel

Kongens Lyngby 2011
IMM-PHD-2011-247

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

Wireless sensor networking is a challenging and emerging technology that will soon become an inevitable part of our modern society. Today wireless sensor networks are broadly used in industrial and civilian application areas including environmental monitoring, surveillance tasks, healthcare applications, home automation, and traffic control.

The challenges for research in this area are due to the unique features of wireless sensor devices such as low processing power and associated low energy. On top of this, wireless sensor networks need secure communication as they operate in open fields or unprotected environments and communicate on broadcasting technology. As a result, such systems have to meet a multitude of quantitative constraints (e.g. timing, power consumption, memory usage, communication bandwidth) as well as security requirements (e.g. authenticity, confidentiality, integrity).

One of the main challenges arise in dealing with the security needs of such systems where it is less likely that absolute security guarantees can be sustained – because of the need to balance security against energy consumption in wireless sensor network standards like ZigBee.

This dissertation builds on existing methods and techniques in different areas and brings them together to create an efficient verification system. The overall ambition is to provide a wide range of powerful techniques for analyzing models with quantitative and qualitative security information.

We stated a new approach that first verifies low level security *protocol*s in a

qualitative manner and guarantees absolute security, and then takes these verified protocols as actions of *scenario*s to be verified in a quantitative manner. Working on the emerging ZigBee wireless sensor networks, we used probabilistic verification that can return probabilistic results with respect to the trade–off between security and performance.

In this sense, we have extended various existing ideas and also proposed new ideas to improve verification. Especially in the problem of key update, we believe we have contributed to the solution for not only wireless sensor networks but also many other types of systems that require key updates. Besides we produced automated tools that were intended to demonstrate what kind of tools can developed on different purposes and application domains.

Resumé

Trådløse sensor netværk er en udfordrende og ny teknologi, der snart vil blive en uundgåelig del af vores moderne samfund. I dag anvendes trådløse sensor netværk bredt i industrielle og civile anvendelsesområder, herunder miljøovervågning, tilsynsopgaver, sundhedsprodukter, home automation, og trafik kontrol.

Udfordringerne for forskningen på dette område skyldes de unikke egenskabber ved trådløse sensor enheder, såsom lav processorkraft og tilhørende lavt energiforbrug. Yderligere er trådløse sensor netværk nødt til at understøtte sikker kommunikation, da de opererer i åbne områder eller ubeskyttede miljøer og kommunikerer vha. radio-teknologi. Som en følge heraf er disse systemer nødt til at opfylde en lang række af de kvantitative begrænsninger (f.eks timing, strømforbrug, hukommelsesforbrug, kommunikation båndbredde) så vels som sikkerhedsmæssige krav (f.eks autentifikation, fortrolighed, integritet).

En af hoved udfordringerne er at opnå det rette niveau af sikkerhed i sådanne systemer, hvor det er usandsynligt, at absolutte sikkerheds garantier kan opretholdes – på grund af behovet for at afbalancere sikkerhed mod energiforbruget i trådløse sensor netværk standarder som ZigBee.

Denne afhandling bygger på eksisterende metoder og teknikker inden for forskellige områder og bringer dem sammen med henblik på at skabe et effektivt verifikationssystem. Den overordnede ambition er at give en bred vifte af kraftfulde teknikker til at analysere modeller med kvantitative og kvalitative sikkerhedsoplysninger.

Vi præsenterer en ny tilgang, der først verificerer lav-niveau sikkerheds *pro-*

tokoller på en kvalitativ måde og garanterer absolut sikkerhed, og derefter tager disse verificerede protokoller som aktioner i *scenarier*, der derefter verificeres kvantitativt. Med udgangspunkt i de nye ZigBee trådløse sensor netværk, benytter vi probabilistisk verifikation – og dermed får vi et probabilistisk indblik i et trade-off mellem sikkerhed og ydeevne.

Vi har således udvidet forskellige eksisterende ideer og foreslået nye ideer til forbedring af verifikationen. Specielt mener vi at vi for problemet med nøgle-opdatering har bidraget til en løsning for ikke blot trådløse sensor netværk, men ogsåmange andre typer af systemer, der kræver nøgle-opdateringer. Derudover har vi produceret automatiserede værktøjer, der har til formål at vise, hvad slags værktøjer der kan udvikles for forskellige formål og anvendelsesområder.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in Computer Science.

The Ph.D study has been carried out under the supervision of Professor Hanne Riis Nielson and Professor Flemming Nielson in the period of September 2007 to January 2011 (excluding the leave of absence period from January 2010 to May 2010).

Most of the work behind this dissertation has been carried out independently and I take full responsibility for its contents. A part of the scientific work in this thesis is based on our published work in [YNN08, YNN09a, YNN09b, YNN10a, YNN11a] with my two supervisors as co-authors. Another part is based on our published work in [YNN$^+$10b, YNN11c] in collaboration with Marta Kwiatkowska and Matthias Fruth from Oxford University. In addition, a part from this thesis is under submission to a conference [YNN11b].

Lyngby, January 2011

Ender Yüksel

Acknowledgements

First and foremost, my thanks go to my supervisors, Hanne Riis Nielson and Flemming Nielson, without whose support, guidance and enthusiasm this work would never have been completed. I consider myself very lucky to be working with them since my master thesis, and I am grateful to them for providing me with the opportunities to work on important research projects and teach stimulating courses.

I would like to thank Marta Kwiatkowska, Matthias Fruth, Dave Parker, Gethin Norman, and all other members of Quantitative Analysis and Verification at Oxford University for their guidance and support during my stay in Oxford and since then.

I would like to thank Gavin Lowe from Oxford University for good advice and interesting discussions on my work with security protocols.

I would also like to thank Robert Cragie from ZigBee Alliance, for his generous technical help in my work on ZigBee.

Thanks must go to Bo Friis Nielsen, Luz Esparza, and Kebin Zeng from Mathematical Statistics group at DTU, and all members of the MT-LAB project for their close collaboration and comments on various parts of my work.

I am also indebted to MT-LAB and SENSORIA projects, together with FIRST and ITMAN PhD schools since I was engaged to and supported by these bodies during my PhD studies.

I would like to thank Mehmet Bülent Örencik from Technical University of Istanbul who has been my role model and inspiration to continue for an academic carrieer.

I want to thank current and former members of the Language-Based Technology group at DTU: Alejandro M. Hernandez, Carroline D.P.K. Ramli, Christian W. Probst, Christoffer R. Nielsen, Eva Bing, Fan Yang, Fuyuan Zhang, Han Gao, Henrik Pilegaard, Jose N.C. Quaresma, Jörg Kreiker, Lijun Zhang, Marian S. Adler, Matthieu S.B. Queva, Michael J.A. Smith, Michal T. Terepata, Nataliya Skrypnyuk, Piotr Filipiuk, Sebastian Nanz, Sebastian A. Mödersheim, Ye Zhang for creating a friendly and stimulating working environment.

I am grateful to the members of my thesis assessment committee, Fabio Martinelli, Jan Madsen, and Stephen Gilmore for accepting to read and review this thesis and for their valuable comments.

Special thanks go to my colleagues on the board of DTU PhD Association, especially Zaza Nadja Lee Hansen, Lirong Yang, Ilka Hoof, Peter Larsen, Arshad Saleem, Yu Chen and Yifan Hu, together with whom we have carried out significant and at the same time exciting tasks.

I would also like to thank Michael Reibel Boesen from DTU Informatics for helping me with Danish in this thesis.

Finally and most importantly, I would like to thank my family and my fiancée Sanem for their never–ending love and support even from miles away. I would like to devote this thesis to them, and hope that they would forgive me for stealing the time for beloved ones and spend it for research instead.

Kongens Lyngby, January 2011

Ender Yüksel

Contents

Chapter 1

Introduction

Problems worthy of attack
prove their worth
by hitting back

Problems, Piet Hein

Information technologies has passed through a rapid sequence of phases since the emergence of computers. Until late 1980s, information was processed in large *mainframe* computers that used tape drives. Then came the era of *personal computers*, which was propagated with the formation of computer *networks*. The trend towards miniaturization continued with the introduction of *embedded systems* which enabled information processing systems to be embedded into enclosing products. In contrast to general–purpose computers, embedded systems are dedicated to specific tasks and the examples of such systems range from portable devices to large stationary installations.

Following the raise of low–powered *wireless sensor network*s (**WSN**), there has been a shift on the emphasis of the information systems. In embedded systems, the emphasis is on computational elements. However, the link between the computational elements and physical elements is missing. This led to a new era in information technologies: *cyber–physical systems* (**CPS**). These systems

integrate computing and communication capabilities with monitoring and *control* of entities in the physical world [CAS08], though they are often viewed as networked embedded systems. Due to its scientific and technological importance, research on CPS is supported by governmental authorities such as the National Science Foundation of the United States of America [NSF10]. Considering the emerging and already prominent CPS technology, we focused our work on security of WSNs which constitute a key element of CPS technology [RLSS10].

WSNs have unique characteristics that require novel security considerations. For example, geographic distribution of sensor devices may allow a physical attack that ends up with physical capture of the devices and extraction of the secret keying material. Another example can be given in data aggregation, an essential paradigm in WSNs where the idea is combining data coming from different resources and saving energy by eliminating redundancy and reducing transmissions. However, routing algorithm of such a system may reveal the devices where the sensed data is aggregated thus allowing an attacker to attack to a more mission–critical device in the network [AK96, DHHM05].

Ensuring security properties such as confidentiality and authentication in a networked system is mainly achieved by small distributed programs called *security protocols*. Even though security protocols rely on cryptographic primitives, they are tremendously error-prone. However, identifying the errors is not trivial even for moderate size protocols. For these reasons, the use of *formal methods* – a combination of a mathematical model of a system and certain requirements, together with an effective procedure for determining whether the system satisfies the requirements – was initiated to verify security protocols. Following this approach, there have been a large variety of formalisms to reason about the correctness of security protocols and formal methods proved successful at discovering flaws with existing protocols [Mea03].

On the other hand, the use of formal methods is still relatively rare in industry which leads to severe flaws in even contemporary communication standards (we will present a real case that we discovered, throughout the thesis). In a very early industrial report, the need for improved integration of techniques based on formal methods has been indicated as an open call to researchers [CGR92]. Over the time, these techniques matured slowly but steadily and embraced a wide perspective including modelling, verification and analysis of computer systems. Nevertheless, the efforts at easing the transition of this technology to a broader user base had been fairly limited.

An important aspect of formal verification is *tool support* which is necessary for stimulating the use of formal methods. While numerous tools are developed and adapted, there is no magical tool that takes a system and completes all the

phases from modelling to verification considering all necessary properties.

We believe that we can achieve an efficient verification system that brings qualitative and quantitative analyses together for the security of communication technologies by using different powerful formalisms in different contexts. Towards achieving this goal, two significant approaches based on formal methods are investigated in this dissertation. As we have mentioned before, we have chosen a challenging environment to work on: WSNs, in particular ZigBee sensor networks – a type of WSN that is less likely to sustain security guarantees due to the need for balancing security against energy consumption. We make use of static program analysis as a method for qualitative security analysis and stochastic model checking as a method for quantitative security analysis. We worked towards establishing a framework that consists of two phases linked together:

- *analysing protocols,* in the sense that qualitative security properties are verified using appropriate methods. In this phase, absolute security – namely complete guarantee of a security property being satisfied – is necessary.
- *analysing scenarios,* where each scenario contains a wise combination of qualitatively verified protocols. In this phase, absolute security is often less important than quantifiable trade-offs between security and other aspects such as performance.

The thesis of the dissertation is:

> *Starting with a lower level of abstraction and applying a qualitative analysis on protocols, then integrating these formally verified protocols as components of scenarios in a higher level of abstraction followed by a quantitative analysis, we can achieve an efficient analysis scheme on information and communication systems.*

Extending the verification to cover *scenario*s in addition to *protocol*s is a key component of our approach. As an example, classical verification of a security protocol is generally based on certain assumptions such as perfect cryptography and proper protection of keying material. However, compromise of a cryptographic key would cause serious hazards even though running protocols are formally verified. Analysis of scenarios provides invaluable information that, for example, assists in finding out which protocols to use and when to change the cryptographic keys in order to prevent attacks from the unknown environment.

Extending protocol verification with quantitative analysis leads to more versatile and efficient analyses. In addition to qualitative security properties such as confidentiality and integrity, we can analyse properties such as timeliness and energy efficiency. Such qualitative properties are getting more important when we work on environments with strict resource constraints such as sensor networks. In this thesis, we aim at making steps towards balancing security against energy consumption. We see a huge potential in applying and tailoring static program analysis and stochastic model checking.

The remainder of this chapter is organized as follows: we present details about the motivation behind this work in Section 1.1, the background on the studied subjects in Section 1.2, the related work in the literature in Section 1.3, and the structure of the dissertation together with an overview on the contributions in Section 1.4.

1.1 Motivation

In this section, we would like to express the problem we chose to attack reminiscent of a rather high level overview. Together with a running story in a user's perspective, we would like to explain why we find the problem we work on interesting and important.

In the near future tiny devices will keep track of our energy consumption and CO_2 emission. They will constantly measure the outdoor temperature and control the air conditioning of our homes. They will be essential in providing the security we want: they will control the access to our homes at the same time as they monitor the smoke and fire detectors in the various rooms. The devices also enter the medical sector where they for example will monitor the health of patients with chronic diseases. The number of application areas seems unlimited.

The devices typically have one or more sensors and a microprocessor with low power consumption. They communicate with one another in wireless networks. A single network may contain just a handful of devices and up to several hundred devices. Typically the network will change over time. We may want to add new devices to the network or we may want to replace existing devices to get more and even better services. A device may also leave the network because it has run out of battery power or because it has lost the connection to the other devices. The network contains a control unit that keeps track of the devices that are currently in the network.

Our modern society will depend on the correct operation of these devices. We cannot accept that the metering of our energy consumption is erroneous. We cannot accept that the security of our homes is jeopardized because these devices do not function properly. We cannot accept that our patients are left on their own because of ill functioning equipment. It is essential that the network is secure – meaning that only devices that have properly registered with the control unit can influence the behaviour of the network.

To achieve this, the devices communicate with one another using cryptographic protocols. Due to limited resources and low power consumption requirements, often it will be the case that all the devices in the network share a common cryptographic key. The messages being exchanged between the devices are then encrypted using this key meaning that all other devices knowing the key can understand the message but no one else. It is the job of the control unit to make sure that all the devices currently in the network know the shared cryptographic key.

Obviously, we have to make sure that the devices and the network itself are preserving certain security properties in addition to a guaranteed correct operation. Following the good practices, we have to verify such properties *before* implementation.

The initial motivation of this thesis is to develop improved methods for verification of security properties of such devices and networks. Second motivation is to facilitate the development and usage of automated tools that are empowered by the formal methods in verification.

1.2 Background

In this section, we briefly present contextual background in a structural manner. Starting with the wireless networks, and continuing with the notions of security protocols and system verification, we will gradually describe background on verification of security protocols in wireless sensor networks. The relation between the aforementioned topics – from this point of view – can be visualized as in Fig. 1.1.

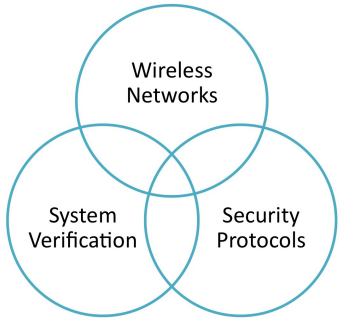


Figure 1.1: Intersection of three main themes.

1.2.1 Wireless Networking

Wireless networking refers to a broad topic that in essence associated with communication networks that use electromagnetic waves – such as radio waves – as carrier and thus provides greater flexibility and convenience compared to wired networks.

A common classification of the wireless networks is done by the range or the area that is covered by the wireless network. Instead of going through details, we will locate the position of ZigBee in wireless networking area using a top-down approach.

- *Wireless Wide Area Networks* provide communication links across metropolitan, regional, or national boundaries by using technologies such as Universal Mobile Telecommunications System, General Packet Radio Service, and 3G to carry voice and data traffic.
- *Wireless Metropolitan Area Networks* are a type of wireless network that connects several Wireless Local Area Networks. A good example for such networks is specified by the WiMAX standard which is built on the IEEE 802.16 standard and preserves connection in a whole city.
- *Wireless Local Area Networks* enable users to establish connection in a local area setting (e.g. inside a building) and provide connection to wider networks such as internet. These type of networks are widely used on a worldwide scale, and Wi-Fi is a well-known technology certification that belongs to WLANs which is based on IEEE 802.11 standard.
- Finally, **Wireless Personal Area Networks (WPAN)** connect network devices within personal area, which is a low cost and short range

type of connection. Bluetooth and ZigBee are both examples of WPANs, based on the same Medium Access Control (MAC) layer family i.e. IEEE 802.15 standard.

ZigBee is at the same time a *wireless sensor network* (WSN) standard, in terms of a classification based on the type of the devices that form the network. A WSN is a network that is formed by a large number of sensor devices. A sensor device is equipped with at least one sensor that detects physical occurrences such as light, heat, motion, or sound.

WSNs are used in many different application areas including automation, monitoring, security, entertainment, and asset tracking. Many of these applications require large number of sensor devices, hence to limit the costs WSN devices have severe resource constraints. These constraints are mainly in terms of computation, memory, and energy. Therefore, security is difficult to achieve, and many well-known methods and approaches become infeasible.

At this point we would like to mention the relation between WSN and CPS. A CPS is generally composed by a set of networked agents, including sensors, actuators, control processing units, and communication devices [CAS08]. In Fig. 1.2 a sample CPS is sketched where a corresponds to an actuator, s corresponds to a sensor, as corresponds to a device with both actuator and sensor, and c being a controller.

While some forms of CPS are already in use, the widespread growth of wireless embedded sensors and actuators is stimulating several new applications – in areas such as medical devices, autonomous vehicles, and smart structures – and increasing the role of existing ones – such as Supervisory Control and Data Acquisition (SCADA) systems.

WSN is one of the key technologies that enable the concept of CPS. Besides, common applications of CPS typically fall under WSNs and autonomous systems.

1.2.2 Security Protocols

A security protocol is a protocol that is used for performing security functions and generally incorporates cryptographic algorithms. The security protocols are widely used for securing the data communication in application level. Those protocols are commonly used for data confidentiality, data integrity, security key

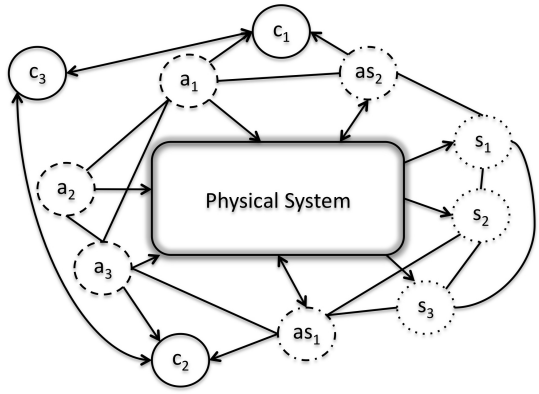


Figure 1.2: A cyber-physical system composed of actuators, sensors, and controllers.

establishment, security key exchange, entity authentication, message authentication, non-repudiation, etc.

Security protocols generally make use of cryptography, so that a virtual secure channel can be established to provide secure communication over insecure media. Cryptography requires cryptographic keys to be established and distributed among the sides of the communication, and such a sequence of message exchanges for key establishment and distribution is a good example of a security protocol.

As we mentioned, security protocols are usually executed in insecure media where malicious users or software can be present. The adversaries are capable of performing many different types of attacks, making it complex to design sound security protocols. Even cryptography cannot save the protocol in most of the situations, which is one of the reasons of security protocols being so error prone. Security protocols are desired to maintain certain security properties. If these security properties cannot be preserved, certain flaws are likely to take place. Those flaws will cause serious attacks in the real implementations. Therefore, both design and verification of the protocols are very important.

1.2.2.1 Security Protocols for Wireless Networks

The intersection of two main themes that we visualized in Fig. 1.1, wireless networks and security protocols, is an important ingredient of this thesis.

In present, there are many security technologies that were developed for the wired networks. However, wireless networks have different characteristics and face more challenges than the wired networks. The main challenge is the medium being reachable to anybody within the transmission range. There is no wire to be hidden inside protections and no physical access restriction. Having the necessary antenna and assuming no security precaution is taken, an attacker can easily interfere with a wireless network. Thinking the other way round, a legitimate user can unintentionally connect to a malicious wireless network that fakes the legitimate one. This so-called *rogue access point* problem doesn't exist in wired networks where a physical plug that is known to be trustworthy is used to connect the network. Another important challenge is the computational and power consumption constraints of wireless networking devices. Encryption is the main way of providing security, whereas it is often a very expensive operation in terms of computation and energy consumption.

At this point, we have to introduce the basic threat types that wireless networks are likely to face. As we have mentioned above, an adversary can sniff the traffic and capture the communication. This is called *eavesdropping* and may cause problems with the help of traffic analysis and cryptoanalysis. In the first case, the adversary may learn information that is plaintext or can be decrypted by the knowledge of adversary. In the second case, the adversary may learn a collection of partial encrypted information and apply cryptoanalysis to predict or decrypt the eavesdropped information. The adversary can also inject messages to the wireless network using sufficient equipment. This is called *message injection*, and may lead to replay attacks. Message injection can also be used to improve eavesdropping by injecting suitable messages that require certain kind of responses to be sent by the victims. In addition, *message deletion* is also possible by causing errors (e.g. checksum errors, errors due to noise) that will make the receiver drop the packet. As we mentioned before, establishing a rogue access point is another treat that will cause the leakage of confidential data.

From WSN security perspective, the use of computationally inexpensive cryptographic techniques has been proposed to ensure confidentiality and authentication [WLSC06]. The complications in these two issues are mainly due to the energy constraints imposed on sensors. In addition, each of the sensor devices in such a network is subject to physical capture, hence such a system must tolerate the compromise of sensors and their security keys. Rather than providing all–or–nothing guarantees about security, there is a need for providing probabilistic guarantees with respect to compromise. Defining models and metrics in this line is an important challenge.

1.2.3 System Verification

In networking area, system verification is in many cases seen as an activity that involves testing and simulation. However, these two methods can not provide the necessary guarantees for systems, such as being flawless or functioning correctly. Therefore, we focus on *formal verification* and refer to the formal methods when we refer to verification.

Formal verification of system correctness depends on the use of mathematical logic. A system can be seen as a mathematical object with well-defined, although possibly complex and intuitively incomprehensible behavior. Mathematical logic can be used to describe precisely what constitutes correct behavior. This makes it possible to mathematically establish that the system behavior conforms to a correctness specification.

Verification of a system requires the formal specification of the system, so that certain properties of the specification can be proved. Such a proof can be made by either human-directed or automated proving. The former requires a hand-written mathematical proof therefore it is bound to a high level of mathematical sophistication. The latter is the one that we are interested in. Producing proofs of properties of systems by automatic means fall into three general categories:

Theorem proving. The correctness of a system is determined by properties of a mathematical theory, using deductive methods. Then, these properties are proved using automatic tools such as theorem provers and proof checkers. As a real life example, this method is used in [BMP03] to verify the SET protocol and in [BDM98] for the automatic train operating system METEOR of the (first) driverless metro-line in Paris.

State exploration / Model checking. A protocol is modelled as a finite-state system and then the verification is evaluated by exploring each state in the protocol and reporting if the protocol enters a state that violates certain properties. A number of model checkers and state exploration methods have been applied to the security protocols as well. Murphi is an early and a well-known example of this group [MMS97, SS98].

Program analysis. An indispensable technique of language-based security which has successfully detected errors in protocols [BBD$^+$03, BBD$^+$05]. Control flow analysis is used to perform an over-approximation of the possible variable bindings and message exchanges. Constructing reference monitor semantics, it is possible to know whether the properties to be validated are violated or not.

1.2.4 Verification of Security Protocols

Above we have briefly explained the background for three main themes that we visualized in Fig. 1.1. We have also touched the intersection of wireless networks and security protocols. In this section, we proceed to the intersection of all these themes. In other words, we continue with the verification of security protocols in wireless networking setting.

Security is a context-sensitive measure in the wireless networks realm. When referring to security we need to identify the properties that we are mentioning. The essential **qualitative properties** to be satisfied in our context, which are sometimes referred to as CIA in the literature (e.g. ITSEC [Com91] – a successor of Common Criteria), are listed below:

Confidentiality. A protocol that ensures confidentiality prevents the disclosure of transmitted data to unauthorized parties, such that only the intended receiver is able to read the confidential data. This is mostly established using cryptography.

Integrity. Messages cannot be changed by any malicious user when data integrity is offered. Modification, insertion, deletion, or replay of transmitted data is detected. Hashing is a well known solution for integrity. In addition, there are some other properties related to integrity such as *non-repudiation*.

Authenticity. Communication over a protocol that offers authenticity means that principals are communicating with the exact principals they believe to be communicating with. To be authenticated means it is ensured that principals are actually who they say they are. Authentication properties have been discussed in many different levels of abstraction. The authentication property studied in [BBD+05] describes authentication at the level of the individual messages used in communication. The idea is to be sure that the messages always have the intended destination and origin, no matter how an attacker interferes with communication.

The abovementioned properties of security protocols are formally verified on an abstract level. In the following paragraphs we will briefly describe the key approaches in this area.

One of the earliest protocol analysis approaches was developed by Dolev and Yao [DY81], who developed a formal model that allowed multiple executions of a protocol running concurrently and including an attacker that could read, modify, and destroy the messages. This work influenced a major amount of the future developments in this area. As we have mentioned in the previous section, inductive theorem proving, state exploration, and program analysis techniques

were generally applied to Dolev-Yao model or its variants.

Later, a different approach which was based on belief logic received attention of the community [BAN90]. It consisted of modal operators describing relations between principals and data, possible beliefs such as owner of a key or sender of a message, and a set of inference rules.

Afterwards different approaches such as type checking came into play, where message and channels were assigned different types and flaws were identified as type violations [Aba99]. Since then many researchers followed this line which offered handling certain classes of *infinte* systems in contrast with model checking.

However, there are also **quantitative properties** which were not taken into account in the early verification methods. For instance, the approaches for the qualitative security properties above do not take *time* into account (except some further developments for the verification of mainly *time-stamps* e.g. [DG04]). This ignorance is not because security protocols are time–insensitive, it is just because ignoring such aspects simplify verification.

Continuing with quantitative temporal information example, an early approach defined a temporal logic to reason about security protocols [Syv93]. Later, use of time automata was suggested for modelling timeout and retransmission. Thus model checkers could be extended to support time-sensitive protocols such as in [BCP09].

Quantitative analysis in a security context has also been used in quantifying security threats such as denial of service. An approach to be noted at this point is studying the effectiveness of countermeasures, which can be applied in potential protocol implementations [BKPA09]. Probabilistic verification tools are heavily used in similar analyses.

1.3 Related Work

In this section, we present some of the related work that address similar objectives with us. In Section 1.3.1, we present related work on automated verification tools. In Section 1.3.2, we continue with the related work on key update. In Section 1.3.3, we map the rest of the related work to various parts in the thesis.

1.3.1 Automated Verification Tools

A major part of this thesis is about automatic verification tools. In literature and in practice, a large number of automated tools are available for protocol verification.

ProVerif [Bla01] and CryptoVerif [Bla08] are two automated tools that verify security protocols in formal models, and computational models, respectively. ProVerif is based on the representation of protocols by Horn clauses, and using approximation it can handle unbounded number of messages with unbounded message space. A disadvantage is the possible presence of false positives in the verification results which is due to approximation. ProVerif can verify confidentiality and authentication properties, in the presence of so called Dolev-Yao attacker. CryptoVerif, on the other hand, is an automatic protocol prover that can also evaluate the probability of success of an attack against the protocol as a function of the probability of breaking each cryptographic primitive and of the number of sessions.

SATMC [AC08] is a SAT–based model checker that verifies security protocols. It generates Boolean formulae that are fed into a SAT–solver. The idea here is building a propositional formula from a description of a protocol in a multi-set rewriting formalism, thus reducing the protocol verification problem to a satisfiability problem of a propositional formula.

Theorem prover Isabelle [NPW02] has been successfully used for the interactive verification in various areas, including protocol verification [Bel07, Pau98]. Similar methods deriving a formal theory that faithfully represents the protocol being analysed and prove one or more theorems (that correspond to properties to be verified) in that theory can be illustrated as [Sch98, BNP02]. Even though theorem provers partially automate the proving process, a certain amount of human intervention is still needed and termination is not always guaranteed.

Tools based on state exploration methods systematically explore the states of the model of the protocol of interest – a finite state model – and look for violations of certain security properties. Brutus [CJM00], FDR/Casper [Low97], OFMC [BMV05], and STA [BB02] make a not necessarily exhaustive list of examples for such tools.

1.3.2 Key Update

Another major part of contribution in this thesis is about key update. We define key update to be the event of revoking valid security key in a network and establishing a new key. We assume a setting where the network consists of resource-constrained devices such as wireless sensor networks. Thus we assume that symmetric cryptography is used, and the necessary keying material is the network key. In other words, all the devices in the network are sharing the same symmetric key. We also assume that the network has a controller, a rather stronger device, that is responsible for key management such as the trust center in ZigBee.

The need for key update is actually related to the risk of key compromise. While key compromise might take place by a device leaving the network (or being removed) with a valid network key, or a message that contains an unprotected keying material, or a brute force attack, or any other reason, a serious source of compromise in wireless sensor networks is physical attacks. Assuming that sensor nodes are tamper-proof often turns out not to be true [AK96]. Based on their real experiments, Deng et al. [DHHM05] showed that it is possible to obtain copies of all the memory and data of a Mica2 mote in tens of seconds, or minutes, after a node is captured, given the proper level of experience and tools.

In the literature, our setting resembles to group key rekeying. Below we will be classifying the related work into three categories.

First category is the *group key* theme. [GL10] proposes the key to be updated when a malicious node is detected, or rekeying timer expires. [PST01, BHRM09] propose a protocol that focuses on performance, e.g. reducing number of rekeying messages. [CGPM05] focuses on how to establish node revocation. [ASDO10] is a study on robustness property, it focuses in key loss and recovery.

Second category which is very closely related to our developments is *rekeying*. The foundational work of Wallner et al. [WHA99] defines a strict and costly policy: update the key in each membership changes. [PJ10] is focusing on the costs of rekeying (i.e. reducing the number of messages), and trying to propose a more efficient protocol than LKH which is the wide-known protocol in multicast group rekeying. Their proposed protocol requires rekeying whenever group membership changes or when a key of a member is compromised. [UBLC10] is an interesting proposal that suggests changing the encryption key depending on the energy level as a one-time key. However using one key per message is not feasible for resource-limited networks.

The last category is *sensor network security specifications* where key update is considered. MiniSec [LMPG07] proposes a balance between energy consumption and security, inspired by TinySec and ZigBee. SPINS [PST+02] only claims that using a master key and then deriving network keys from this key would be good since a key compromise will not affect the master key. However, it does not consider rekeying. LEAP+ [ZSJ03] is a well known key management protocol for sensor networks. It considers rekeying, and proposes methods for different key types, however still not going further then advising periodical key updates.

1.3.3 Related Work In Various Chapters

In addition to automated verification tools and key update topics above, we have several other topics that we contributed on. However, remaining work related to these topics are distributed throughout the thesis. In this section, we will present a mapping from topics to corresponding related work.

ZigBee security is a backbone of this thesis, and we reserved almost a whole chapter for that. The related work on ZigBee security is spread over Chapter 2.

Qualitative security analysis and static program analysis being our one of the two key analysis domains has its related work in Chapter 3. The other key analysis domain, quantitive security analysis and stochastic model checking enjoys related work in a fairly big preliminaries chapter: Chapter 4.

1.4 Outline

In this section, we provide an outline of the dissertation including the structure and the contributions. We start by presenting a visual overview of the components that build the thesis in Fig. 1.3. Each small box having a numbered caption in Fig. 1.3 is actually representing a *chapter*, and each framing box numbered with roman numerals and captioned in boldface fonts is representing a *part* that contains multiple chapters.

In addition to the separation by parts, we have a separation by analysis domains. As you can see in Fig. 1.3, the left hand side including Part I is the qualitative domain whereas the right hand side including Part II and Part III is the quantitative domain. Furthermore, in Automated Tools for Analyses part, one of the chapters is totally related to quantitative analysis and other chapter takes place on both of the domains.

As you can see in the organization of domains we have a bias towards the quantitative analysis, such that a stand-alone part of Case Studies and a stand-alone chapter in Part IV are on this subject. This does not mean that we have a discrimination between two domains, it is merely a result of our assumption of targeted readers being experienced with qualitative analysis in particular program analysis.

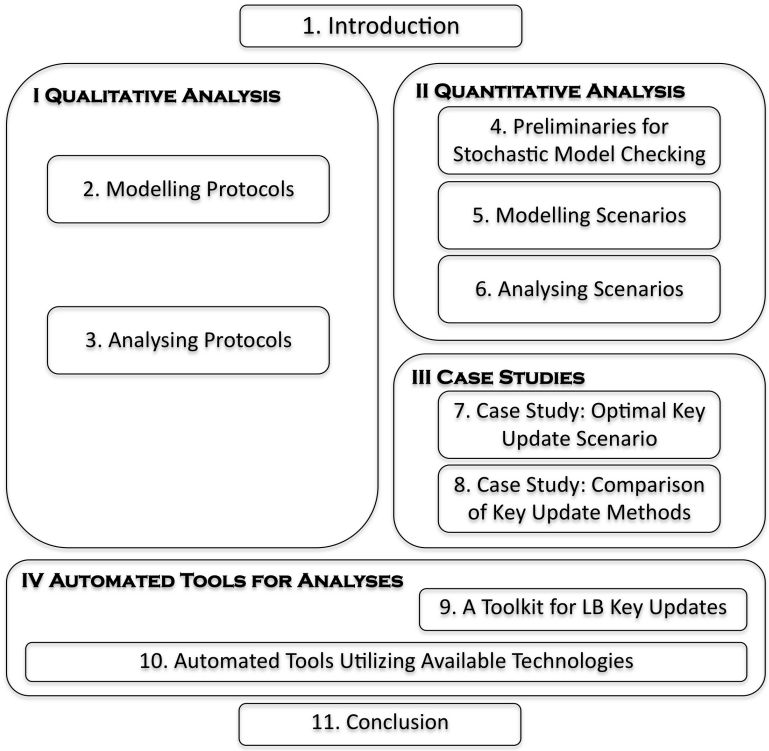


Figure 1.3: Outline of the dissertation

1.4.1 Organization of the Dissertation and Overview of the Contribution

In this section, we present the organization of the dissertation. For each part that we have presented in Fig. 1.3, we have included a short description that links them to related published work, and a definition for each included chapter. These chapter definitions are composed of two parts: a short description of the chapter, and a short summary of the contribution of that chapter.

Part I: Qualitative Analysis
The first part of the dissertation is mainly concerned with qualitative analysis of security protocols. This part is based on the papers [YNN08, YNN09b, YNN10a].

- **Chapter 2** introduces the WSN standard that we will use as a running example throughout this dissertation: *ZigBee*. Focusing on the security sublayer of ZigBee, Chapter 2 presents how to proceed from standard specification to security procedures, and eventually security protocols.

  Contribution. As a contribution to both ZigBee development and protocol verification communities, we clarified the security sublayer of ZigBee, spotted important changes by comparing with the previous specification, and derived protocol narrations from the specification which will be useful for both implementations and security analyses. We also identified a critical gap in the specification on key updates.

- **Chapter 3** introduces our approach for protocol analysis together with a real–world example where a significant flow in a security protocol is discovered and fixed.

  Contribution. We presented the usage of static program analysis technique, and we applied this technique on the contributions of Chapter 2. As a result we discovered a critical flaw in the key establishment protocol of ZigBee, which is a real protocol that is being used in ZigBee implementations. In addition, we proposed a fix that corrects the protocol and showed another usage of this analysis technique also in verifying the fix.

Part II: Quantitative Analysis
The second part of the dissertation is mainly concerned with quantitative analysis of security scenarios. This part is based on the papers [YNN09a, YNN$^+$10b].

- **Chapter 4** introduces the preliminaries for stochastic model checking technique, covering the necessary foundations that we will need in the following chapters.

- **Chapter 5** defines the problem that we will attack using the approach presented in the previous chapter. In addition, it presents our work towards developing a formal model that is both realistic, scalable, and convenient for stochastic model checking.

  Contribution. We produced a very compact and scalable model that reduces model checking costs in terms of time and memory, and allows model checking of large WSNs.

- **Chapter 6** explains how stochastic model checking technique can be applied to a part of a wireless network standard's security sublayer such that we can reason about security properties and get quantitative answers to our questions.

  Contribution. We presented a methodology for determining optimal key update policies and security parameters, considering the security needs for realistic application scenarios. The contribution can be widely applicable to other types of networks, as well.

Part III: Case Studies The third part of the dissertation consists of case studies that utilize quantitative analysis. This part is based on the papers [YNN11a, YNN11b, YNN11c].

- **Chapter 7** demonstrates how to derive advice from quantitative analysis results in three different case studies focusing on different application domains of WSNs.

  Contribution. We improved our formal models, so that we get more insights in the analyses. We proposed new key update methods that considers the nature of WSNs, but can also be used in various networking schemes. We presented an application of quantitative analysis in the security of WSNs, focusing on three different application domains.

- **Chapter 8** presents an application on different key update methods, which allows us to asses the methods under changing network conditions.

  Contribution. We constructed an analysis to observe how different key update methods adapt to new environmental conditions in a network. We evaluated the methods and presented the results as a set of guidelines for network designers. We also proposed an adaptive mechanism to make a more efficient use of key update methods, considering power consumption.

Part IV: Automated Tools for Analyses The fourth part of the dissertation is describing the tools that we designed and implemented to demonstrate our selected developments in the previous chapters in an automated way.

- **Chapter 9** presents a toolkit that implements maximum risk analysis and transient analysis on a certain key update method. In addition, it demonstrates how to construct models in an analytical way, instead of a compositional way.

  Contribution. We developed an analytical model in a mathematical perspective, rather than a computer science perspective. In addition, we

implemented a toolkit that computes the maximum risk in transient security analysis, and applies a smart transient analysis which covers only the necessary time period for security analysis.

- **Chapter 10** presents automated tools designed and developed for solving some of the problems discussed in the thesis.

  Contribution. We have provided automated tools to be used by not only security and verification experts but also network designers or even people with a very limited knowledge on the topic. We have implemented a push–button technology tool that implements all the analysis in determining optimum key update method and parameter values for a given network. We have also implemented a tool that bridges the gap between protocol narrations and LySa models, and also automates qualitative verification using LySa engine.

Part I

Qualitative Analysis

Chapter 2

Modelling Protocols

In Chapter 1, we classified wireless networks by their ranges and we defined WPANs as the shortest range wireless networks. In this chapter, we introduce ZigBee which is a type of WPAN and also a wireless sensor network. We describe the security structure of ZigBee in details, in order to form basis for our developments in the forthcoming chapters.

As a solid contribution to both ZigBee and protocol verification communities, we derive the protocol narrations from the specification which will be useful for both implementations and security analyses.

We named this chapter as *modelling protocols*, yet we do not mean a formal model. Instead, we model in a security protocol notation. We will make use of the protocols we modelled in this chapter by converting them into formal models and applying a qualitative analysis in the following chapter.

We survey ZigBee security in Section 2.1, where we define almost all the ZigBee related terms that we will use in our developments. In Section 2.2, we define the security protocols using the security procedures. In Section 2.3, we present the vulnerabilities of ZigBee.

2.1 From Specification To Procedures

Literally, a *specification* is an explicit set of requirements to be satisfied by a material, product, or service. In the field of network standards, a specification often specifies one or more layers of a network. In this thesis, we are working on ZigBee networks, therefore we will start with the ZigBee Specification.

Again, literally a *procedure* is a specified series of actions which have to be executed in the same manner, in order to *always* obtain the same result under the same circumstances. As we focus on security, we will work towards extracting security procedures out of the specification. We will describe the security procedures in details, after completing this section.

In this section, we present a high level self-contained overview and explain the key security components of the latest ZigBee specification, *ZigBee–2007* [Zig08d]. We explain the important points in indispensable protocols, algorithms, and computations. In fact, we are constructing a knowledge base for our developments in the following chapters.

2.1.1 ZigBee Wireless Sensor Networks

In this section, we first introduce ZigBee wireless sensor networks and then discuss the reason of working on this wireless networking standard and using it as an extensive running example for the whole thesis.

2.1.1.1 Introducing ZigBee Networks

ZigBee is a fairly new but promising WPAN standard for wireless networks that consist of devices with very low resource requirements. The name *ZigBee* is derived from the *zigzag* dance of the honey bees that guide their hive members to flowers. Metaphorically, simplistic ZigBee devices work together to tackle complex tasks.

ZigBee can be used in many different WSN applications such as wireless switches, electrical meters, consumer electronics equipment, industrial control, embedded sensing, medical data collection, intruder warning, home automation, and many more. In the early considerations, ZigBee was merely seen as a technology that could make life easier with various applications, a few of which were mentioned in [ZLA06] such as “configuring a home network so that the light intensity is

lowered automatically when you turn on the TV, and the TV will mute itself when the phone rings". Thus security was not considered as a critical factor in the implementations, and the trade-off between security and performance (might be related to power consumption, cost, speed, etc.) was not even mentioned in most cases. Indeed, ZigBee provides framework for such applications but currently it is wrong to consider ZigBee as a technology for just simple residential applications. In fact today, ZigBee has a growing acceptance across different markets that require high security such as Smart Energy [Zig08c] where wide deployments are in progress in USA, Sweden, Canada, Korea, Australia, and China. Such markets have very little tolerance in security flaws and therefore ZigBee security sublayer is recently enhanced to provide high security on this still low-rate wireless sensor networking standard.

ZigBee-2007 has different *application profile*s. An application profile is a common application level language for exchanging data in a given application domain. It specifies both a possible collection of devices, and a set of messages used by the devices in communication. Current ZigBee public application profiles are *Commercial Building Automation*, *Home Automation*, *Personal Home and Hospital Care*, *Smart Energy*, *Telecom Applications*, and *Wireless Sensor Applications*.

2.1.1.2 What Makes ZigBee Interesting

In the context of protocol verification, there are several reasons to choose a wireless networking standard like ZigBee. First of all, ZigBee is a fairly new networking standard and thus very little research is conducted. Therefore we can try many new ideas, improve many issues that are untouched, and pinpoint new research problems for future research. In this thesis, indeed we have applied both quantitative and qualitative analyses making use of the open problems of this new networking standard, and along the way we succeeded to discover a security flaw, fix the flawed protocol, proposed new key update schemes, and decision algorithms for key update strategies.

Choosing a wireless standard is meaningful in the context of security, since wireless networks face a lot more challenges than the wired networks. However, another important point in selecting ZigBee to work on is the *low-rate* nature that makes implementing and preserving security requirements much more challenging. Besides, as a sensor network standard, ZigBee devices have long battery life expectations which is a tough restraint on the operations that consume battery.

Following the trends in wireless communication, we found out that ZigBee is be-

coming a more security-critical sensor network protocol suit since the application areas are enlarging from less secure lighting/heating sensing to Smart Energy where security is a *must* requirement. This fact forced the security designers to strengthen the security sublayer, so that many new security components are added lately. Obviously, the improvements raise the importance of verification because adding new and more security components does not always mean that the level of security is raised.

As we mentioned, ZigBee is designed for low-rate wireless networks which have low costs, low power consumption, low range and low bandwidth. In parallel with this, the devices that will operate in ZigBee networks have limited resources in terms of memory, processor, storage, power, etc. Therefore implementing the security guarantees is a great challenge and the verification of the security properties is of paramount importance.

2.1.2 ZigBee Security Overview

In this section, we describe general security principles and present an overview of ZigBee security. Throughout this thesis, ZigBee refers to the ZigBee-2007 Specification [Zig08d] unless otherwise stated.

A *feature set* is an agreement of configuration parameters, network settings and policies. ZigBee-2007 contains two feature sets that interoperate network-wise: *ZigBee* and *ZigBee* PRO. The ZigBee PRO feature set has two security modes (Standard Security and High Security, to be explained in Section 2.1.3.3), and three types of security keys (Network Key, Link Key, and Master Key, to be explained in Section 2.1.3.1), whereas the ZigBee feature set has only Standard Security mode and two types of security keys (Network Key, Application Link Key). Both feature sets use symmetric encryption, the *Advanced Encryption Standard* (**AES**-128) [Fed01], and apply authentication/encryption on Network and Application layers.

Related to the device types, we have two different classifications: hardware and logical. The hardware device type distinguishes the type of the hardware platform and it may be either *Full Function Device* (**FFD**), or *Reduced Function Device* (**RFD**) according to the relevant IEEE standard that we will explain in a moment. A logical device type distinguishes devices in a specific network and it may be *Coordinator*, *Router*, or *End Device* in a ZigBee network.

Before we start talking about ZigBee security architecture, we need to mention that IEEE 802.15.4 standard [IEE03] defines low level specifications of ZigBee, as

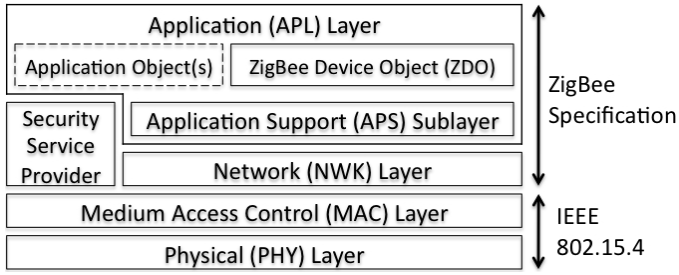


Figure 2.1: ZigBee Stack Architecture

well as many other wireless sensor network standards[1]. Thus, ZigBee guarantees to have a solid specification for both physical radio frequency (RF) transmission and medium access.

The IEEE 802.15.4 provides reliable communication between a device and its neighbours, addressing critical issues such as collision avoidance and improving efficiencies in the communication. It provides the interface to the physical transmission medium, and handles assembly and decomposition of data packets.

ZigBee is built on a Physical layer (**PHY**) and a Medium Access Control layer (**MAC**), both defined by the IEEE 802.15.4-2003 standard. The PHY layer can operate in two separate frequency ranges: *lower* 868MHz (European) and 915MHz (United States, Australia, etc.) and *higher* 2.4 GHz (worldwide). The MAC layer controls access to the radio channel using a *carrier sense multiple access with collision avoidance* (CSMA-CA) mechanism. Upon this structure, ZigBee builds the Network layer (**NWK**) and the Application layer (**APL**) which consists of the Application Support sublayer (**APS**) and the ZigBee Device Object (**ZDO**). Fig. 2.1 shows the ZigBee stack architecture, including the end manufacturer defined part in dashed box. This kind of visualization is useful for locating where we are in this study. We focus on the Security Service Provider part of the ZigBee Specification, which interacts with the NWK and APS layers.

The IEEE 802.15.4 allows two types of network topologies: *Star* and *Peer–to–peer*. In star topology, each device talk directly to the Coordinator, whereas in peer–to–peer topology each device can talk to any other device in its range. ZigBee, making use of this, allows three topologies: *Star*, *Tree*, and *Mesh*.

[1]A common mistake that we observed in the literature is, ZigBee uses the 2003 version of this standard, namely IEEE 802.15.4-2003, but not the more recent IEEE 802.15.4-2006 which actually superseded the former.

Mesh topology enables high levels of reliability and scalability by providing more than one path through the network. Tree topology utilizes a hybrid of star and mesh topologies by combining the benefits of both high levels of reliability and support for battery-powered nodes.

In this thesis, we will assume star topology since star networks are very common and they provide very long battery life operation.

Below we discuss the basic security principles in ZigBee. We first introduce the security service provider, then explain the open trust model, and then the architectural design choices.

2.1.2.1 The Security Service Provider

The Security Services Specification is a chapter in the ZigBee specification which specifies the security services available within the ZigBee stack. These services include methods for key establishment, key transport, frame protection and device management. As shown in Fig. 2.1, ZigBee provides security mechanisms for NWK and APS layers. Each layer is responsible for securing their frames. In addition, the APS sublayer provides services for security relationship establishment and maintenance whereas ZDO manages the security policies and the configuration of ZigBee devices. Throughout this chapter, we present the overall security architecture and the security mechanisms in different layers.

2.1.2.2 Open Trust Model

The ZigBee security architecture depends on assumptions such as safekeeping of the symmetric keys, proper implementation of the cryptographic mechanisms and the associated security policies involved. Besides, ZigBee assumes that different applications using the same radio are not logically separated (*e.g. by firewall*). Also, a device cannot verify whether cryptographic separation between different applications/layers on another device is properly implemented. Therefore, it must be assumed that separate applications using the same radio trust each other, which means that there is no cryptographic task separation. In addition, the lower layers (i.e. APS, NWK, MAC) are fully accessible to the applications. As a result, ZigBee has the notion of an *open trust model* that is derived from these assumptions. This model states that the security services only protect the interfaces between different devices, but not the interfaces between different layers nor the different applications on the same device. Here, protection means cryptographic protection, and for the separation of the inter-

faces between the different stack layers on the same device non-cryptographic mechanisms should be employed in the designs. As stated in the specification, this model allows the reuse of the same keying material among different layers on the same ZigBee device. The model also allows end-to-end security to be realized on a device-to-device basis instead of layer-to-layer basis.

2.1.2.3 Architectural Design Choices

In this subsection we focus on the design choices listed in the current ZigBee Specification. The main point in these choices is that any malevolent device should not be able to use the network to transport frames across the network without permission.

1. *The layer that originates a frame is responsible for initially securing it.* A simple example from the specification: If a NWK command frame needs protection, then NWK layer security must be used.

2. *If protection from theft of service is required, then NWK layer security shall be used for all frames.* Only a device that joined the network and authenticated (i.e. received the active network key) will be able to communicate using the network. However, it is not possible to apply NWK layer security between a router and an (joined but) unauthenticated device.

3. *Security can be based on the reuse of keys by each layer.* This is for reducing the storage costs.

4. *End-to-end security is enabled, based on a shared key between only two devices.* This actually limits the trust requirement to only two devices communicating.

5. *The security level used by all devices in a given network, and by all layers of a device shall be the same.* If an application requires more security, then it shall form its own separate network with a higher security level.

The ZigBee specification states that other decisions such as key update/expire, counter overflow, loss of synchronization, error conditions arising from securing frames, etc. should be included in the application profiles and must be addressed correctly in the real implementations. However, this way of delegating critical issues to real implementations instead of defining in the specification creates security hazards [YNN08].

Table 2.1: Key Acquirement Schemes

Method \ Key Type	**NK**	**MK**	**LK**
Key Transport	Available	Available	Available
Key Establishment	—	—	Available
Pre-installation	Available	Available	Available

2.1.3 A Closer Look At The Security-Related Components

In this section, we introduce three main concepts in ZigBee security: *security keys*, *trust center*, and *security modes*. We will make use of these three concepts very often in not only this but also following chapters.

2.1.3.1 Security Keys

ZigBee devices use 128-bit symmetric encryption keys to provide security amongst a network. A *Link Key* (**LK)** is shared by two ZigBee devices and it is used to secure unicast communication between APL peer entities. LK is used as the basis of the security services (i.e. key transport, authentication) in *High Security* mode. A *Network Key* (**NK**) is shared amongst all devices in a network and it is used to secure broadcast communications in a network. NK is used as the basis of the security services (i.e. authentication, frame security) in *Standard Security* mode. A *Master Key* (**MK**) is used to establish a key and it is shared pairwise between two ZigBee devices. The intended recipient is always aware of the key type that is used in frame protection.

The security keys can be acquired in different ways depending on their types as shown in Table 2.1. *Key Transport* is the case that the Trust Center of the network sends the key to the device. *Key Establishment* is the method that is used to establish a pairwise key (*i.e. LK*) between two devices. Note that for this method, a pre-shared key (*i.e. MK*) is required between two devices. *Pre-installation* is the case that the device acquires the key before joining the network.

In ZigBee-2007, NKs have two different types: standard (SNK), and high-security (HSNK). It is stated that NK distribution and initialization of the Frame Counters (*See Section 2.1.4.1*) depend on the type of the NK, but in both cases the messages are secured in the same way. The availability of the security keys to the different layers and different security modes are given in Table 2.2. Some key types are optional for some security modes and they are indicated with "(O)" in the table. In addition, all ZigBee layers must share the

Table 2.2: Availability of Key Types

Keys	Layers		Modes	
	NWK	**APL**	**SS**	**HS**
NK	Available	Available	Available	Available
MK	—	Available	—	Available (O)
LK	—	Available	Available (O)	Available (O)

Table 2.3: Derived Key Types (`0x`: Hexadecimal)

Key Type	**Computation**	**Explanation**
Key-Transport Key	$\text{HMAC}(\texttt{0x00})_{LK}$	Protects transported NKs
Key-Load Key	$\text{HMAC}(\texttt{0x02})_{LK}$	Protects transported MKs and LKs
Data Key	LK	Equal to the LK

active NK and associated incoming/outgoing frame counters.

In order to avoid reuse of keys across different security services, it is possible to derive different keys from the LK. Uncorrelated keys can be derived using a one-way function so that execution of different security protocols can be logically separated. Three types of secret keys can be derived from a LK as listed in Table 2.3. The derivation of these keys, except Data Key, requires computation of a *Keyed-Hashing for Message Authentication Code* (**HMAC**) [Fed02]. All the derived keys must share the associated frame counters.

2.1.3.2 Trust Center

In each secure ZigBee network, there exists a unique Trust Center (**TC**) application which is trusted by all the devices in the network. TC distributes keys as part of network and end-to-end application configuration management. In high-security (commercial) applications devices are using MK, whereas in low-security (residential) applications devices are using NK to initiate secure communication with TC. In parallel with the key acquirement schemes in Table 2.1, MK and NK can be obtained by either pre-installation or a kind of key transport which is called *in-band unsecured key transport*. Certainly, the latter option is not tolerable in vulnerable situations. The interaction between a ZigBee device and the TC for different purposes is given in Table 2.4.

As we mentioned before, there are three logical device types in a ZigBee network: *coordinator*, *router*, and *end device*. TC is not a device type, but an application. In a ZigBee network, the ZigBee Coordinator configures the security level (which

Table 2.4: Key Distribution

Purpose	**Device receives**	**Via**
Trust Management	Initial MK or Active NK	Unsecured Key Transport
Network Management	Initial active NK and updated NKs	Secured KeyTransport
Configuration	MK or LKs	Secured Key Transport

can also be *unsecured*). The ZigBee Coordinator also configures the TC of the network, which is by default the ZigBee Coordinator itself.

2.1.3.3 Security Modes

TC can be configured to operate in either Standard Security mode (**SS**) or High Security mode (**HS**). In SS mode, which is designed for the residential applications, the TC is required to maintain the SNK and control the policies of network admittance. In HS mode, which is designed for the commercial applications, the TC is required to maintain a list of all the devices in the network (this is actually optional in SS mode), all the relevant keys (MKs, LKs, HSNKs) and control the policies of network admittance. As a result, the required memory of the TC grows with the number of the devices in the network in HS mode, but not in SS mode. In addition, the implementation of the Symmetric-Key Key Exchange (*See Section 2.1.5.2*) and the Mutual Entity Authentication (*See Section 2.1.5.3*) protocols are mandatory in HS mode. Right now, among the application profiles mentioned in the beginning of Section 2.1.2, only the Commercial Building Automation profile uses the HS mode.

2.1.4 Security of Layers

In this section, we explain the security architecture of ZigBee in a layer-wise fashion. We start with the network layer security in Section 2.1.4.1, and continue with the application layer security in Section 2.1.4.2. Then we explain the security of the application support sublayer in Section 2.1.4.3, which is very important even though it is a *sublayer*.

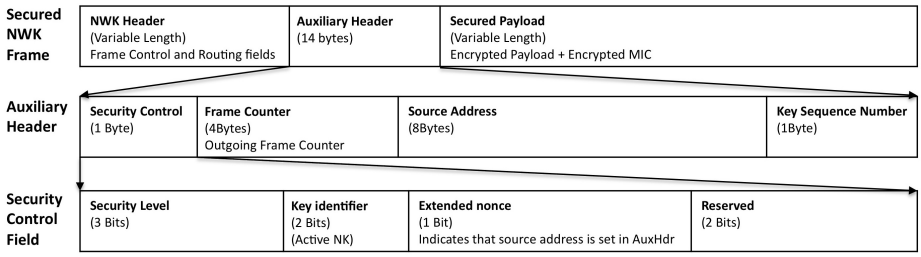


Figure 2.2: Secured ZigBee NWK Frame

2.1.4.1 Network Layer Security

The Network layer (NWK) provides functionality to ensure correct operation of the MAC layer and also provides suitable service interface to the APL layer. When a NWK layer frame needs to be secured, the NWK layer secures it by using AES encryption/authentication in the *Enhanced Counter with CBC-MAC* (CCM*) mode of operation (*see Section 2.1.5.1*). The NWK layer processes outgoing/incoming frames in order to securely transmit/receive them. The upper layers control the security processing operations by setting up the security keys, frame counters and the security level.

The structure of a secured NWK frame is given in Fig. 2.2. A secured NWK frame is a NWK frame that has an Auxiliary Header (**AuxHdr**) and this is indicated in the *Frame Control* field of the NWK Header. AuxHdr includes the *Frame Counter*, which has the purpose of providing frame freshness and preventing processing of duplicate frames. An important point here is that the word "secured" does not necessarily mean that encryption is applied (*i.e. in the case of integrity-only protection*). Therefore, a Secured Payload does not need to have an encrypted payload.

Security Levels. The *Security Level* field in the Security Control part of the AuxHdr indicates which security level is applied. The level can be:

- **None:** No security at all, in terms of confidentiality and integrity.
- **MIC**[2]**:** Integrity protection only, with three different Message Integrity Code (**MIC**) lengths:

[2] *MIC* is actually the same as message authentication code (MAC). However, MIC is preferred by ZigBee (and IEEE) to distinguish between another MAC: Medium Access Control Layer.

  - **MIC-32:** Message Integrity Code of 32-bits.
  - **MIC-64:** Message Integrity Code of 64-bits.
  - **MIC-128:** Message Integrity Code of 128-bits.
- **ENC:** Encryption only.
- **ENC-MIC:** Both encryption and integrity protection with three different MIC lengths:
  - **ENC-MIC-32:** Message Integrity Code of 32-bits.
  - **ENC-MIC-64:** Message Integrity Code of 64-bits.
  - **ENC-MIC-128:** Message Integrity Code of 128-bits.

The MIC is computed using the NWK header, the AuxHdr and the (encrypted) payload.

Table 2.5: NWK Frame Security (||: Concatenation)

Processing Outgoing Frames	**Processing Incoming Frames**
1. Retrieve active NK, outgoing frame counter, key sequence number and security level from the NWK Layer Information Base (**NIB**). 2. Set AuxHdr using the parameters in Step 1. 3. Execute CCM* mode encryption and authentication with the parameters: length of MIC (obtained from security level), NK, and CCM* Nonce (CCM* Nonce uses the values from AuxHdr and constructed as "Source Address ‖ Frame Counter ‖ Security Control". 4. Construct the outgoing frame, depending on the encryption requirement, as in Fig. 2.2. 5. Increase outgoing frame counter in NIB. 6. Set the *Security Level* subfield of AuxHdr as "000".	1. Determine security level from NIB and overwrite it to the *Security Level* subfield. 2. Determine the key sequence number, sender address, and the *Received Frame Count* from AuxHdr. 3. Obtain the corresponding (matching the key sequence number) security material from NIB. If *Received Frame Count* is less than *Frame Count* then FAIL. 4. Execute CCM* mode decryption and authentication with the same parameters as in outgoing frames. 5. Set Frame Count to "Received Frame Count + 1". Store Frame counter and the Sender Address in the NIB.

The processing of the incoming and outgoing frames are given in Table 2.5. An interesting security precaution here is hiding the security level in the last step

of Outgoing Frames Processing. Although the rationale behind this action is not defined in the specification, it is clear that it is not a significant protection since there are only eight choices (as listed above) [YNN08].

2.1.4.2 Application Layer Security

As shown in Fig. 2.1, the Application layer (APL) is composed of the APS sublayer, the ZDO and manufacturer defined application objects. A maximum of 240 distinct application objects can be defined. ZDO is responsible for initializing APS, NWK, Security Service Provider, and assembling the information from applications. ZDOs are applications that employ NWK and APS primitives to implement ZigBee End Devices, ZigBee Routers and ZigBee Coordinators. When an APL layer frame needs to be secured, the APS sublayer handles security. Therefore, APL security actually covers APS sublayer security which is explained in Section 2.1.4.3.

2.1.4.3 Application Support Sublayer Security

The Application Support sublayer (APS) provides an interface between the NWK and the APL layers through a general set of services (for use of ZDO and Application Objects) provided by APS data and management entities. The APS sublayer processes outgoing/incoming frames in order to securely transmit/receive the frames and establish/manage the cryptographic keys. The upper layers issue primitives to APS sublayer to use its services. APS Layer Security includes the following services: *Establish Key*, *Transport Key*, *Update Device*, *Remove Device*, *Request Key*, *Switch Key*, *Entity Authentication*, and *Permissions Configuration Table*. Below we explain the current usage of these services, keeping in mind that they can be extended in the future.

The **Establish Key** service is the mechanism for establishing a LK between two ZigBee devices. In ZigBee-2007, Symmetric-Key Key Exchange (SKKE, to be explained in Section 2.1.5.2) is the method for key establishment and MK is the trust information used in key establishment. Recently, Public-Key Key Establishment (PKKE) is also enabled in a ZigBee application profile [Zig08c].

The **Transport Key** service provides secure or insecure means to transport a NK, LK, or MK.

The **Update Device** service provides secure means for a ZigBee Router to inform the TC that a device changed its status (*i.e. joined or left the network*).

The **Remove Device** service provides secure means for the TC to inform a ZigBee Router that one of his children should be removed.

The **Request Key** service provides secure means for a device to request the *active NK* or the *end-to-end application MK* from another device (*i.e. TC*).

The **Switch Key** service provides secure means for TC to inform a device to switch to the alternate NK.

The **Entity Authentication** service, *which was not present in the previous ZigBee specification, ZigBee-2006* [Zig06], provides authenticity between two devices based on a shared secret (*i.e. NK*).

The **Permission Configuration Table** (**PCT**) stores the information of which devices have authorization to perform which commands. In addition, PCT determines whether security based on LK is required or not. Maintaining a PCT is optional.

The services of the APS sublayer are issued in *APS Command Frame*s. The structure of the APS Command Frames is given in Fig. 2.3. The first two fields of all the frames, the Frame Counter and the APS Counter, form the *APS Header*. The remaining fields form the *APS Payload*, whose first field *APS Command Identifier* indicates the type of the command frame (SKKE, Transport-Key, Update-Device, Remove-Device, Request-Key, Switch-Key, Entity Authentication, etc.), as shown in Fig. 2.3.

There are some important points regarding the APS command frames in Fig. 2.3. All *Key Establishment* (*e.g. SKKE*) command frames are sent unsecured. The *Status* field of the *Update-Device* command indicates the security mode (SS/HS), the security of the frame (Secured/Unsecured), and the type of the join (Join/Rejoin); unless the device is leaving. *Partner Address* field of the *Request-Key* command is not present when the key type is NK or TCLK (note that TCLK means LK of the TC).

We have mentioned in Table 2.5 that the NKs are stored in the NWK Layer Information Base (**NIB**). Likewise, the MK/LK pairs and the relevant information is stored in the APS Layer Information Base (**AIB**).

The processing of the incoming and outgoing frames in APS layer is given in Table 2.6. Note that the security level is hidden the same way as in the NWK layer frame security.

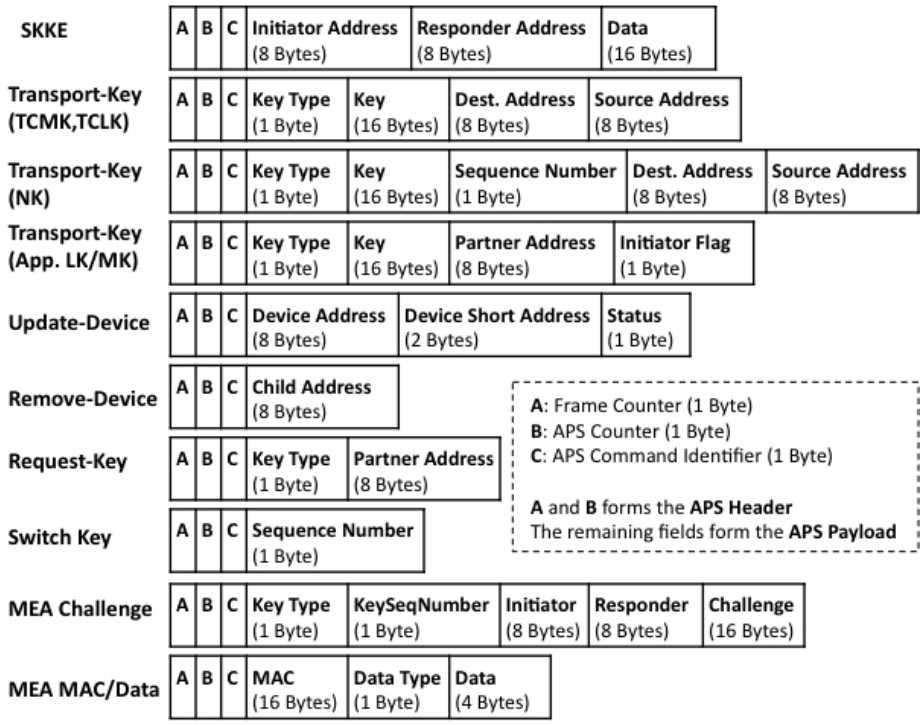


Figure 2.3: APS Command Frames

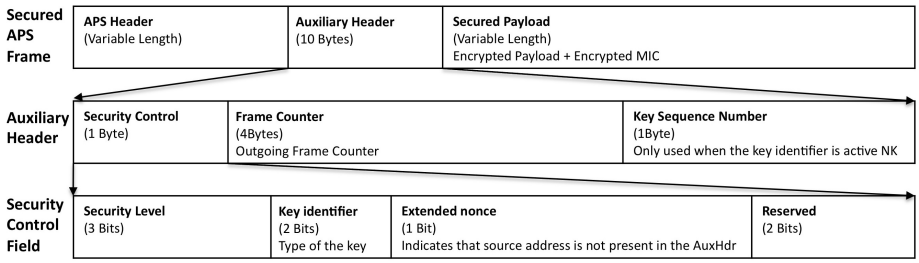


Figure 2.4: Secured ZigBee APS Frame

2.1.5 Advanced Concepts in ZigBee Security

In this section, we will introduce some advanced concepts in ZigBee security that both brings remarkable improvements to ZigBee specification and also relevant to our developments. We start with a special encryption mode, and continue

Table 2.6: APS Frame Security (||: Concatenation)

Processing Outgoing Frames	**Processing Incoming Frames**
1. Obtain the security material (from NIB or AIB), and the key identifier. If the key identifier is *active NK*, then either APS or NWK layer will apply security (not both of them). 2. Extract the outgoing frame counter (and the key sequence number if the key identifier is active NK) from the security material. 3. Obtain the security level from NIB. Set the AuxHdr (using all these parameters) as in Fig. 2.4. 4. Execute CCM* mode encryption and authentication, construct the CCM* nonce as "Source Address ‖ Frame Counter ‖ Security Control" (using the values from AuxHdr). 5. Construct the outgoing frame, depending on the encryption requirement, as in Fig. 2.4. 6. Increment and store the outgoing frame counter. 7. Set the *Security Level* subfield of AuxHdr as "000".	1. Determine the sequence number, key identifier and received frame counter value from the AuxHdr. 2. Obtain the appropriate security material from NIB or AIB depending on the key identifier. If *Received Frame Count* is less than *Frame Count* then FAIL. 3. Determine the security level from NIB and overwrite it to the *Security Level* subfield. 4. Execute CCM* mode decryption and authentication with the same parameters as in outgoing frames. 5. Unsecured APS frame will be constructed using the output of CCM*. 6. Set and store the Frame Count as "Received Frame Count + 1".

with two protocols that are inherited to the ZigBee standard.

2.1.5.1 CCM* Mode of Operation

Enhanced Counter with CBC-MAC (**CCM***) is a generic encryption and authentication block cipher mode which is defined for use with only block ciphers having 128-bit block size. The AES-CCM* mode of operation is an extension of the AES-CCM mode that is used in IEEE 802.14.5-2003 [IEE03] and provides capabilities for authentication, encryption, or both. Securing a NWK or an APS frame is actually based on AES-CCM* mode of operation in a particular security level.

2.1.5.2 Symmetric-Key Key Establishment Protocol

In the Symmetric-Key Key Establishment (**SKKE**) protocol, an initiator device U establishes a LK (or LK_{UV}) with a responder device V using a shared secret MK (or MK_{UV}). This protocol is actually inherited from another standard, ANSI X9.63:2001 [Ame01]. As we mentioned in Table 2.1, MK may either be pre-installed or transported from the TC. We present the SKKE protocol with computational details in Table 2.7. In the first two messages, the devices exchange their 16-byte challenges. In the last two messages, the devices exchange the data they have computed using the challenges and the device identities. Note that a *device identity* is the unique 64-bit device address, and ***kdf*** (defined in [Ame01]) is the key derivation function that takes two parameters: the shared secret bit string, and the length of the keying data to be generated. After verifying that they received the correct values, they use another value as the LK that both of them can compute.

All the SKKE messages use the frame format as shown in Fig. 2.3. The *Data* field in a SKKE frame stores the value of either a challenge or a MAC (not the MAC layer, but the Message Authentication Code) tag.

2.1.5.3 Mutual Entity Authentication Protocol

In the Mutual Entity Authentication (**MEA**) protocol, an initiator device U and a responder device V mutually authenticate each other based on a secret key (NK). The devices authenticate each other by using random challenges with responses based on a NK. We present the MEA protocol with computational details in Table 2.8. In the first two messages, the devices exchange their challenges. Note that, **OFC** stands for the Outgoing Frame Counter of the device. In the last two messages, the devices exchange the data they have computed using their information and their frame counter values. They use the frame counter values they received and their previous knowledge to verify that they received the correct values, and thereby authenticate each other.

The MEA protocol uses two different frame formats for the challenge and the data, as shown in Fig. 2.3. The *MAC*, *Data Type*, and *Data* fields have the values field is the *MacTag*, *Frame Counter*, and the field is the *Frame Counter Value*, respectively.

Table 2.7: SKKE Protocol (H: Hash Function, MAC: HMAC Function, ||: Concatenation, 0x: Hexadecimal)

SKKE-1	U→V	QEU
SKKE-2	V→U	QEV
Computation in U		Z = MAC{U‖V‖QEU‖QEV}$_{MK}$ KKeyData = kdf(Z, 256) MacKey = Leftmost 128bits of KKeyData = H(Z‖0x00000001) KeyData = Rightmost 128bits of KKeyData = H(Z‖0x00000002) MacData2 = 0x03‖U‖V‖QEU‖QEV MacTag2 = MAC(MacData2)$_{MacKey}$
SKKE-3	U→V	MacTag2
Verification in V		MacData2 = 0x03‖U‖V‖QEU‖QEV Verify MacTag2 using MacData2
Computation in V		Z = MAC{U‖V‖QEU‖QEV}$_{MK}$ KKeyData = kdf(Z, 256) MacKey = Leftmost 128bits of KKeyData = H(Z‖0x00000001) KeyData = Rightmost 128bits of KKeyData = H(Z‖0x00000002) MacData1 = 0x02‖V‖U‖QEV‖QEU MacTag1 = MAC(MacData1)$_{MacKey}$
SKKE-4	V→U	MacTag1
Verification in U		MacData1 = 0x02‖V‖U‖QEV‖QEU Verify MacTag2 using MacData2
U and V use KeyData as the new LK LK = H(MAC{U‖V‖QEU‖QEV}$_{MK}$‖0x00000002)		

2.2 From Procedures To Protocols

In the beginning of this chapter we have introduced the notions of *specification* and *procedure*, and up until now we have explained the specification that we are interested. We have done this in a way that leads us to the security procedures. A security procedure is applied in certain cases to achieve a defined goal. In this case, it is between ZigBee devices, and are highly related to the services that application layer provides as we have explained in Section 2.1.4.3.

Now, we need to define a *protocol*. Literally, a protocol is a set of guidelines or rules. In our context, a security protocol is used for performing security functions and generally incorporates cryptographic algorithms.

Table 2.8: MEA Protocol (MAC: HMAC Function, ||: Concatenation, 0x: Hexadecimal)

MEA-1	U→V	QEU
MEA-2	V→U	QEV
Computation in U		MacData2 = 0x03‖U‖V‖QEU‖QEV‖OFC_U MacTag2 = MAC{MacData2}$_{NK}$
MEA-3	U→V	MacTag2, OFC_U
Verification in V		MacData2 = 0x03‖U‖V‖QEU‖QEV‖OFC_U Verify MacTag2 using MacData2
Computation in V		MacData1 = 0x02‖V‖U‖QEV‖QEU‖OFC_V MacTag1 = MAC(MacData1)$_{NK}$
MEA-4	V→U	MacTag1, OFC_V
Verification in U		MacData1 = 0x02‖V‖U‖QEV‖QEU‖OFC_V Verify MacTag1 using MacData1

This section presents our first main contribution in this dissertation. After studying the ZigBee specification and the related documentation carefully, we have extracted protocol narrations for each security procedure in ZigBee. This is not a trivial step towards protocol verification, since ZigBee specification chose to narrate procedures in a story-telling context where ambiguities exist and possibility of misunderstanding is very high. For example, authentication procedure is explained in a total of eight pages of highly confusing story-telling. On the other hand, both developers, implementors, and protocol verifiers need protocols that are specified in an unambiguous and well-defined manner. The traditional and simple *Alice-Bob* notation is never used in the ZigBee specification, which would eliminate most of the problems.

We have comprehended the security procedures of ZigBee, and derived corresponding security protocols to be used in both implementations and mostly in verification. We have presented all the protocol narrations we derived in Appendix A, using a notation that we have extended from the classical Alice-Bob notation. Besides, we have filled the fields of the protocol messages with the possible values, where applicable.

In this section, we present a procedural description of how the security services in ZigBee are used. The security procedures consist of *joining a secured network*, *authentication*, *NK update*, *end-to-end application key establishment*, and *network leave*. We tried to visualize the security procedures in a state machine form in Fig. 2.5. The arrows in the figure symbolize the procedures, and the boxes symbolize the states of a ZigBee device in the security context. Note that the procedure *End-to-End (abbreviated as E2E in the figure) Key Establishment* that is used for establishing pairwise keys, is only valid in the HS mode. Also

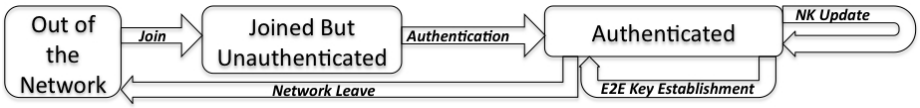


Figure 2.5: Security Procedures

note that it is also possible to change states without using procedures, as in the case of time outs, missed key updates, etc.

2.2.1 Joining A Secured Network

This procedure is applied when a joiner device communicates with a router to join a secure network or when a device in a network has missed a key update and wants to receive the latest NK. Hence, the state of the device is *out of the network* as can bee tracted in Fig. 2.5. The joiner device may begin the procedure by transmitting an unsecured beacon request frame. The joiner device receives beacons from routers and decides which network to join. After that, the joiner device sends an association/rejoin request to the router. The router, knowing the joiner device's address and security capabilities (and in the case of *rejoin* whether the NK was used to secure the rejoin request command), will send an association/rejoin response command to the joiner device. If the joiner device receives a positive association/rejoin response command, the joiner is declared as *joined but unauthenticated* and the next phase shall be the Authentication routine. The protocol narration for the *Join* procedure is given in Appendix A.1.2.

2.2.2 Authentication

A ZigBee device that successfully finished the *joining a secured network* procedure is declared as *joined but unauthenticated* and shall start the *authentication* procedure. If the device is not a router, then after successful completion of the authentication it will be declared as *joined and authenticated*. If the device is a router, then after successful completion of the authentication followed by the initiation of routing operations it will be declared as *joined and authenticated*.

A new feature of the ZigBee-2007 is that, not only the newly joined device but also the neighbouring routers must be authenticated by using the MEA protocol.

Authentication depends on many different parameters such as the security mode

(SS, HS), the presence of a router between TC and the device, the security level (None, ENC, MIC, ENC-MIC), the preconfiguration of the device (Not preconfigured, Preconfigured with NK/MK/LK), profiles, etc. We have visualized the branching of authentication protocols in Fig. 2.6.

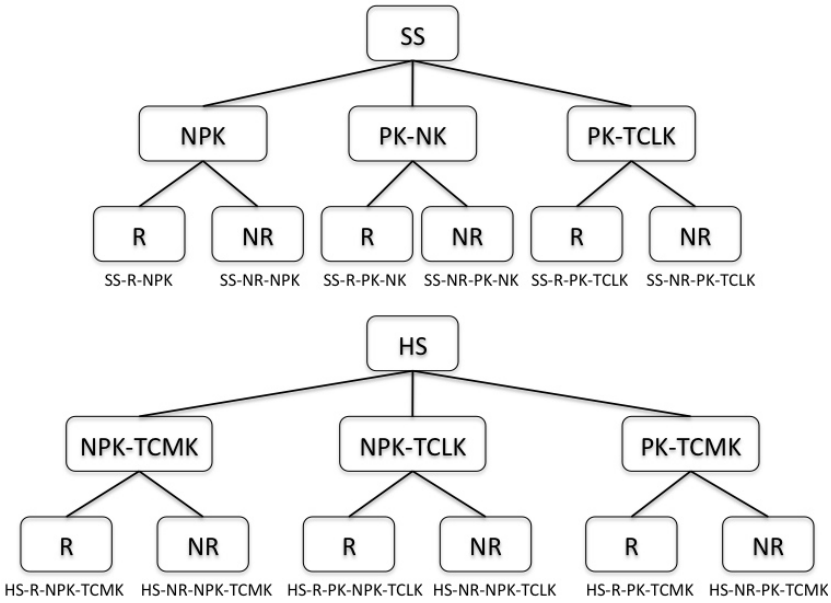


Figure 2.6: Authentication Protocols

In SS mode, there are three main scenarios:

NPK: Device is not preconfigured with a valid key: TC sends active NK to the device in plaintext.

PK-NK: Device is preconfigured with active NK: TC sends fake NK (all zeros) to the device in plaintext.

PK-TCLK: Device is preconfigured with TCLK: TC sends NK to the device, encrypted by TCLK (LK of the TC).

In HS mode, there are two main scenarios:

NPK: Device is not preconfigured with a valid key: According to its configuration, TC sends either TCLK or TCMK in plaintext (*We separated these cases as NPK-TCLK and NPK-TCMK, see Appendix A*). If TC sends TCMK, then it also makes key establishment to derive a TCLK.

PK-TCMK: Device is preconfigured with TCMK: TC establishes TCLK with the device, using key establishment service.

In case of HS, the joiner establishes entity authentication with the router to complete the authentication procedure. In the case of a separate router existing between the TC and the device, the communication between the router and the TC is secured in APS layer using active NK in SS, TCLK in HS. The protocol narrations for the *Authentication* procedure is given in Appendix A.1.3.

2.2.3 NK Update

We have mentioned that, maintaining a list of all the devices in the network is optional in SS but mandatory in HS for a TC. In SS mode, TC broadcasts new SNK to all the devices. In HS mode, TC sends new HSNK to each device using unicast communication.

If the device is capable of storing an alternate NK (*i.e. FFD*), then it will replace its alternate NK with the new key it received. If the device is not capable of storing an alternate NK (*i.e. RFD*), then it will replace its current NK and ignore any Switch-Key command. In any case, all incoming frame counters and the outgoing frame counter of the appropriate NK shall be set to 0. The sequence number of the new key will be the sequence number of the previous key incremented by 1 in mod 256. The protocol narration for *NK Update* procedure is given in Appendix A.1.4.

2.2.4 End-to-End Application Key Establishment

When End-to-End Application Security between two devices is required, the devices will run this procedure which also requires the collabration of the TC. Depending on the configuration of the TC, the devices can either receive an application LK (**appLK**) or an application MK (**appMK**) from TC. In the case of receiving an appMK, it is necessary to establish an appLK afterwards, using the SKKE protocol. The protocol narration for the *End-to-End Application Key Establishment* procedure is given in Appendix A.1.5.

2.2.5 Network Leave

The Network Leave procedure actually works in two different ways: Remove-Device and Device-Leave. In *Remove-Device*, the TC wants a ZigBee device to be removed from the network. After TC informs the router about the situation, the router sends a leave command to the relevant device. In *Device-Leave*, a device decides to leave the network and sends a leave command to the router. Then, the router informs the TC about the situation. In either case, TC shall delete the device from its list (which is optional in SS mode). If TC and the router share a LK, then the messages between the two will be secured with LK, otherwise with NK. The messages between router and the leaving device will be secured by NK. The protocol narration for the *Network Leave* procedure is given in Appendix A.1.6.

2.3 Vulnerabilities

In this section, we explain main vulnerabilities of ZigBee by referring to some of the related work that we found highly relevant. We have not only covered the related work on ZigBee security, but also IEEE 802.15.4 security. The IEEE 802.14.5 standard and its potential security vulnerabilities are important since it is the structure that ZigBee is built on.

In [SW04], Sastry and Wagner pinpoint the problems in IEEE 802.14.5 security and classifies them as: (*1*) Initialization Vector (**IV**) Management Problems, (*2*) Key Management Problems, and (*3*) Integrity Protection Problems. Since the vulnerabilities may be avoided in different levels, the paper [SW04] also classifies the advice for these levels. The advice for application designers include avoiding usage of any security suite that does not have integrity protection (e.g. ENC), and for implementing their own acknowledgement system. The advice for hardware designers include improving Access Control List (ACL) and nonce usage, and eliminating the implementations of the security suites without integrity. Finally, the advice for specification/standard writers include necessary support, requirements and explanations for the vulnerable points which are pinpointed as problems.

In [ZLA06], Zheng et. al present an attack classification based on layers. Jamming, capturing, tampering, exhaustion, collision, and unfairness are the attacks that are possible in the PHY and MAC layers. Routing disruption and resource consumption are the attack types that are possible in the NWK layer. In [ZLA06] some of these attacks are modelled in the network simulation sys-

tem NS-2 [MFF] and the results are presented. In addition, relying heavily on the TC is an important criticism in the paper. Distributed or hierarchical key management schemes are recommended especially for large scale networks.

We believe that *key update* is an important issue in ZigBee networks. When a device is removed or leaving the network, it still knows the security key (i.e. NK). However, updating the NK after each device leave will be costly, whereas not updating the key will be insecure. Therefore, there is a trade-off between security and performance. We have pinpointed this problem in [YNN08].

Burglary protection is an interesting topic in pervasive computing. The attacks on security protocols are not the only threats in ZigBee networks, *theft* is also a serious security threat.

In [PP07], Pedersen and Pagter propose a security policy which is an extension of a more general model called "Resurrecting Duckling" [SA00]. The main idea in [PP07] is to chain the devices in a network or in friendly networks in such a way that a device will only function when it can see all of its friends. This idea is realized with protocols defined for device association and presence verification, and some real life scenarios are presented in the paper.

Chapter 3

Analysing Protocols

Computer networks or simply networks are the main means of information sharing and communication in today's IT infrastructure. Certain protocols are executed to facilitate communication in networks. However, such networks are mostly insecure and the communication needs to be protected against attackers that may influence network traffic and and communication parties that might be either dishonest or compromised by attackers.

Cryptographic security protocols form an essential ingredient of network communications by ensuring secure communication over insecure networks. These protocols use cryptographic primitives to support certain security properties, but ensuring this properties requires a lot more effort. Despite the relatively small size of the security protocols it is very hard to design them correctly, and their analysis is very complicated. One of the most well-known examples is the Needham-Schroeder protocol [NS78], that was proven secure by using Burrows–Abadi–Needham (BAN) logic [BAN90]. Seventeen years later G. Lowe [Low95, Low96], found a flaw by using an automatic tool *Failure–Divergence Refinement* (FDR). The flaw was not detected in the original proof because of different assumptions on the intruder model. The fact that this new attack had escaped the attention of the experts was an indication of the underestimation of the complexity of protocol analysis. This example has shown that protocol analysis is critical for assessing the security of such cryptographic protocols.

In this chapter, we present our approach for protocol analysis together with a real example where we found an important flow in a contemporary wireless sensor network security protocol. We start by modelling protocols using a specific process algebraic formalism called LySA process calculus. We then apply an analysis based on a special program analysis technique called control flow analysis. As a real-world example, we apply this technique to the ZigBee-2007 End-to-End Application Key Establishment Protocol and with the help of the analysis discover an unknown flaw. We suggest a fix for the example protocol, and verify that the fix works by using the same technique.

3.1 An Overview of the Analysis Method

Static program analysis, in essence, examines a program statically, before any attempt of execution. Although the finite amount of resources may limit the information or the answers to important questions, the approximation based approach of static program analysis makes it preferable on the area of protocol analysis. Instead of facing undecidability problem, this technique sacrifices precision and gives approximate answers about a property of a certain program, or a piece of code, or a protocol as in our case. However, the loss of precision does not mean that we are missing the flaws, it merely means that the analysis results may include false positives, such as a bug or a flaw that the program does not contain.

Static program analysis was originally developed for generating codes and optimising compilers [LM69, BE69]. Nevertheless, the analysis technique have recently been directed to the field of security. Encouraging results have been obtained by the use of this approach where safe approximations to the set of values or behaviours arising during protocol runs can be predicted.

Control flow analysis of processes formalised in the LySA process calculus successfully computes an over-approximation of the run-time behaviour of a protocol [BBD$^+$03, BBD$^+$05]. This method is actually the protocol analysis method that we present in this chapter. The roadmap of the analysis method is given in Fig. 3.1, and we will present the steps of this roadmap in the following sections.

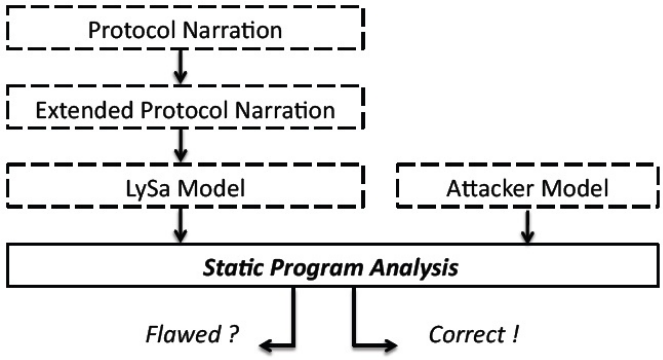


Figure 3.1: The Roadmap of the Analysis

3.2 Modelling in LySa Process Calculus

The first step in the protocol analysis is to formalise the protocol narration into a model that is suitable for the analysis. In our case, we formalize the protocols using the LySA process calculus [BBD+05]. LySA is based on the π-calculus [Mil99] and incorporates cryptographic operations using ideas from the Spi-calculus [AG97]. However, LySA has two different properties compared to spi/π calculus. First, LySA has one global ether, instead of channels. The reason for this difference is that, in usual networking implementations (e.g. ethernet-based, wireless, etc.), anyone can eavesdrop or act as an active attacker which does not correspond to the channel-based communication. The second difference is in the pattern matching usage in the tests of the expressions associated with input and decryption. Although LySA is a very powerful process calculus which also supports asymmetric encryption, digital signatures, etc., in order to make it simple we only illustrate the *symmetric* fragment. The symmetric fragment suffices to prove our claims in the example that we will present the flaw discovery since the protocol is designed for symmetric encryption only. The reader interested in further details including the asymmetric fragment may refer to [BBD+05].

In LySA, we have terms (E) that consist of names (keys, nonces, messages, etc.), variables, and the compositions of them using symmetric encryption. The syntax of terms is shown in Table 3.1. In the case of encryption, the tuples of terms $E_1, \ldots, E_k$ are encrypted under a term E_0 which actually represents an encryption key. Note that an assumption of perfect cryptography is adopted, which means that *decryption with the correct key* is the only inverse function of encryption. The *annotation* inside brackets in the end of encryption will be explained later in this section.

Table 3.1: LySA Terms - Symmetric Fragment

$E ::=$	
x	variable
$\mid \quad n$	name
$\mid \quad \{E_1, \ldots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]$	symmetric encryption

Table 3.2: LySA Processes - Symmetric Fragment

$P ::=$	
0	nil
$\mid \quad P_1 \mid P_2$	parallel comp.
$\mid \quad !P$	replication
$\mid \quad (\nu\, n)\, P$	restriction
$\mid \quad \langle E_1, \ldots, E_k \rangle.P$	output
$\mid \quad (E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$	input
$\mid \quad \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k\}^{\ell}_{E_0}[\mathsf{orig}\,\mathcal{L}\,]\ \mathsf{in}\ P$	symm. decryption

The syntax of the processes (P) which is mostly alike to the polyadic Spi-calculus [AG97] is shown in Table 3.2. At this point, we prefer to skip the syntax for simple ones in the table, but explain the more interested and complicated two: output and input processes. The output process $\langle E_1, \ldots, E_k \rangle.P$ sends the k-tuple $E_1, \ldots, E_k$ to the network and continues as process P. Similarly, the input process $(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$ receives a k-tuple $E'_1, \ldots, E'_k$ and if conditions are satisfied, removes the k-tuple from the network. Here, the input operation uses pattern matching which will only succeed if the prefix of the input message matches the terms specified before the semi-colon. In a simple manner, we can say that for some input E' the input process $(E;x).P$ means that if E' can be separated into two parts such that first part pairwise matches to the values E, then the remaining part of the input will be bound to the variables x. As you can see in Table 3.2, the number of tuples in E' is k so that this is the total number of tuples in E and x. This kind of pattern matching is also used in decryption.

Example 3.1 (Restriction and Output) *The example* LySA *code below is a* ***new*** *(created - restriction) encryption key (K) followed by an* ***output*** *which includes three plaintext elements (A, B, K_A) and an encrypted element ($\{K\}_{K_A}$).*

$$(\nu\, K)\, \langle A, B, K_A, \{K\}_{K_A} \rangle$$

■

Example 3.2 (Input) *The example* LySA *code below is an* ***input*** *that binds*

the last two elements of the input to the variables x_{KA} and x as long as the first two elements are A and B.

$$(A, B; x_{KA}, x)$$

■

Example 3.3 (Decryption) *The example* LySA *code below is a* ***decryption*** *that decrypts the value bound to variable x using the encryption key bound to variable x_{KA} and binds the resulting plaintext value to the variable x_K. Note that this decryption always succeeds without any need of pattern matching, as long as the correct key exists in the receiver.*

$$\mathsf{decrypt}\ x\ \mathsf{as}\ \{;\ x_K\}_{x_{KA}}$$

■

In order to describe the *message authentication* intentions of the protocols, we also have *annotation*s for origin and destination. Encryptions can be annotated with fixed labels called *crypto-point*s that define their positions in the process, and with *assertion*s that specify the origin and destination of encrypted messages. A crypto-point ℓ is an element of some set $\mathcal{C}$ and used when encryptions/decryptions occur. The LySA term for encryption:

$$\{E_1, \ldots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\ \mathcal{L}\,]$$

means that the encryption happened at crypto-point ℓ and the assertion [dest $\mathcal{L}$] means that corresponding (valid) decryption is *to* happen at a crypto-point that belongs to the set $\mathcal{L}$ such that $\mathcal{L} \subseteq \mathcal{C}$. Similarly, in the LySA term for decryption:

$$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k\}^{\ell}_{E_0}[\mathsf{orig}\ \mathcal{L}\,]\ \mathsf{in}\ P$$

[orig $\mathcal{L}$] specifies the crypto-points $\mathcal{L} \subseteq \mathcal{C}$ that E is allowed to have been encrypted.

Example 3.4 (Annotations and Parallel Composition) *The example* LySA *code below is the* ***composition*** *of the three separate parts given in Example 3.1,*

Example 3.2, Example 3.3, and the necessary ***annotations*** *in such a way that now we have two separate processes running in* ***parallel****.*

$$
\begin{array}{ll}
/^*\ a\ ^*/ & (\nu\, K)\, \langle A, B, K_A, \{K\}^{\ell_A}_{K_A}[\mathsf{dest}\,\{\ell_{\mathcal{B}}\}\,]\rangle.0 \\
 & \mid \\
/^*\ b\ ^*/ & (A, B; x_{KA}, x). \\
/^*\ c\ ^*/ & \mathsf{decrypt}\ x\ \mathsf{as}\ \{;\, x_K\}^{\ell_B}_{x_{KA}}[\mathsf{orig}\,\{\ell_{\mathcal{A}}\}\,]\,\mathsf{in}\,.0
\end{array}
$$

The example we constructed step by step is actually the LySa *model of the single-message protocol below:*

$$\textbf{\textit{1. A}} \rightarrow \textbf{\textit{B:}}\ KA,\ \{K\}_{KA}$$

The upper part (line a) of the parallel composition is the code for principal A, and the lower part (lines b and c) is for principal B. In this example, annotations state that the encryption at crypto-point ℓ_A is intended to be decrypted only at ℓ_B. In a corresponding manner, the decryption at ℓ_B should originate from the encryption at ℓ_A. ■

3.2.1 Specifying Protocols in LySa

In the beginning, we have a protocol narration like the one in Table 3.3. The protocol consists of three messages, exchanged between principles A, B, and TC. The notation we have used in Table 3.3 is the well-known Alice-Bob notation which has three parts in each message: *message number*, *message origin and destination*, *message fields*. In this little example, the principal A asks the principle TC to create and send a pairwise key for using in secure communication with another principle B. As a response, TC sends the link key that it has established, *LK*, to both A and B in two separate messages. The communication between A and TC is secured (encrypted) by their pairwise key *KA* (in messages 1 and 2), and similarly the communication between B and TC is secured by their pairwise key *KB* (in message 3).

Table 3.3: Protocol Narration Example

1.	**A→TC:** $\{\text{TC, B}\}_{KA}$
2.	**TC→A:** $\{\text{A, LK}\}_{KA}$
3.	**TC→B:** $\{\text{B, A, LK}\}_{KB}$

Then we extend the narration to specify the internal actions to be performed in principals when receiving those messages. The reason for this kind of extension or conversion is to completely state the actions internal to the principals, which are normally left implicit in the narration of security protocols.

As an example, the extended protocol narration of the sample protocol in Table 3.3 is given in Table 3.4. For each message in the original protocol narration that has message number n, we have a corresponding output message n and a corresponding input message n' in the extended protocol narration. Input message n' presents the variable (those written in *italics*) bindings and necessary checks in the receiver side. If a variable is a ciphertext and the receiver has the correct encryption key, then we have another message (i.e. n'') for each of those variables. In addition, we explicitly write the internal actions as annotations between square brackets, in order to bridge the gap between informal and formal specification of the protocol. Note that when analysing protocols we add an extra message to the end, where a principal attempts to communicate the other through the new shared key, LK. For example, the message

$$\mathbf{1.\ B} \rightarrow \mathbf{A:}\ \{\mathrm{MSG}\}_{LK}$$

does not change the protocol nor bring any (nor bring any additional cost to the implementations), it is just a sample message that will be sent using the new LK and thus it will ease the validation which is done by checking attackers knowledge.

In the next phase, we convert the extended protocol narration into a LySA model. We use the LySA syntax that we explained earlier in this section and configure the necessary settings. As an example, a regular LySA model of the protocol that we have used to demonstrate extended protocol conversion is given in Table 3.5. Further details of specifying protocols in LySA are present in [BBD+05].

3.3 Static Program Analysis

Static Analysis is a formal method that enables the security analysis of cryptographic communication protocols which are modelled as LySA processes. Messages communicated on the network are tracked with the possible values of the variables in the protocol. Besides, the potential violations of the destination/origin annotations are also recorded. The aim of static analysis is to efficiently compute the safe approximations to the behaviour of the models without

Table 3.4: Extended Protocol Narration Example

1. A	$\rightarrow$	: A, TC, $\{\text{TC, B}\}_{KA}$
		[dest TC]
1'.	$\rightarrow$ **TC**	: $x_{initiator}$, x_{TC}, $x_{message}$
		[check x_{TC}=TC]
1".	**TC**	: decrypt $x_{message}$ as $\{x'_{TC}, x_{responder}\}_{KA}$
		[orig x_A][check x'_{TC}=TC]
2. TC	$\rightarrow$	: [new LK]
		TC, $x_{initiator}$, $\{x_{initiator}, \text{LK}\}_{KA}$
		[dest $x_{initiator}$]
2'.	$\rightarrow$ **A**	: y_{TC}, y_A, $y_{message}$
		[check y_{TC}=TC, y_A=A]
2".	**A**	: decrypt $y_{message}$ as $\{y'_A, y_{LK}\}_{KA}$
		[orig TC][check y'_A=A]
3. TC	$\rightarrow$	: TC, $x_{responder}$, $\{x_{responder}, x_{initiator}, \text{LK}\}_{KB}$
		[dest $x_{responder}$]
3'.	$\rightarrow$ **B**	: z_{TC}, z_B, $z_{message}$
		[check z_{TC}=TC, z_B=B]
3".	**B**	: decrypt $z_{message}$ as $\{z'_B, z_{initiator}, z_{LK}\}_{KB}$
		[orig TC][check z'_B=B]
4. B	$\rightarrow$	: [new MSG]
		B, $z_{initiator}$, $\{\text{MSG}\}_{z_{LK}}$
		[dest $z_{initiator}$]
4'.	$\rightarrow$ **A**	: y_B, y''_A, $y_{message2}$
		[check y_B=B, y''_A=A]
4".	**A**	: decrypt $y_{message2}$ as $\{y_{msg}\}_{y_{LK}}$
		[orig B]

actually running them. In Fig. 3.2 we can see the approximation approach. In general, it is impossible to compute the precise answer so we make a choice

Table 3.5: LySa Model Example

	let $X \subseteq \mathbf{N}$ s.t. $\lfloor \mathbf{N} \rfloor = \{1,2,3\}$ in
	$(\nu_{i \in X}\ KA_i)\ (\nu_{j \in X}\ KB_j)$
	$\mid_{i \in X} \mid_{j \in X \cup \{0\}} !$
/* **1** */	$\langle A_i, TC, \{TC, B_j\}_{KA_i} [\text{at } a1_{ij} \text{ dest } \{tc1_{ij}\}\,] \rangle.$
/* **2'** */	$(TC, A_i;\ y_{ij}).$
/* **2"** */	decrypt y_{ij} as $\{A_i;\ xLK_{ij}\}_{KA_i}$ [at $a2_{ij}$ orig $\{tc2_{ij}\}$] in
/* **4'** */	$(B_j, A_i;\ y2_{ij}).$
/* **4"** */	decrypt $y2_{ij}$ as $\{;\ xmsg_{ij}\}_{xLK_{ij}}$ [at $a4_{ij}$ orig $\{b4_{ij}\}$] in 0
	$\mid$
	$\mid_{j \in X} \mid_{i \in X \cup \{0\}} !$
/* **3'** */	$(TC, B_j;\ z_{ij}).$
/* **3"** */	decrypt z_{ij} as $\{B_j, A_i;\ yLK_{ij}\}_{KB_j}$ [at $b3_{ij}$ orig $\{tc3_{ij}\}$] in
/* **4** */	$(\nu\ MSG_{ij})\ \langle B_j, A_i, \{MSG_{ij}\}_{yLK_{ij}} [\text{at } b4_{ij} \text{ dest } \{a4_{ij}\}\,] \rangle.\ 0$
	$\mid$
	$\mid_{i \in X \cup \{0\}} \mid_{j \in X \cup \{0\}} !$
/* **1'** */	$(A_i, TC;\ x_{ij}).$
/* **1"** */	decrypt x_{ij} as $\{TC, B_j;\ \}_{KA_i}$ [at $tc1_{ij}$ orig $\{a1_{ij}\}$] in
/* **2** */	$(\nu\ LK_{ij})\ \langle TC, A_i, \{A_i, LK_{ij}\}_{KA_i}$ [at $tc2_{ij}$ dest $\{a2_{ij}\}\,] \rangle.$
/* **3** */	$\langle TC, B_j, \{B_j, A_i, LK_{ij}\}_{KB_j}$ [at $tc3_{ij}$ dest $\{b3_{ij}\}\,] \rangle.\ 0$

between over-approximation and under-approximation. Static analysis over-approximates the set of possible operations that the LySa process describes. The nature of over-approximation may cause the analysis to investigate a trace which is impossible at all. However, over-approximation is needed to make a safe approximation since under-approximation could miss some traces.

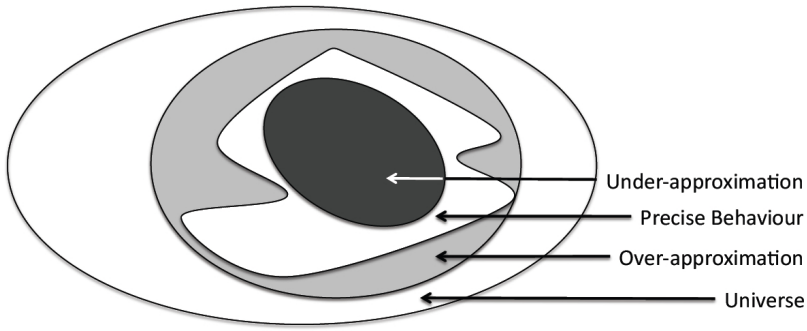


Figure 3.2: Approximation and Precise Behaviour

3.3.1 Analysis Method

The static analysis we use in this study is specified as a Flow Logic [BBD+03, BBD+05], which is based on the control flow analysis and the data flow analysis techniques that allow us to make it fully automatic [NNH99].

Control flow analysis is a program analysis technique that is used to compute approximations of the result of a program execution without running the program. Such an analysis helps us in determining the sets of values that may be generated by communication using a specific protocol, which is beneficial for validating certain security properties. Especially when used in conjunction with a model of possible malicious activity (i.e. attacker), the analysis provides a safe approximation of all events that may happen.

Flow Logic is a notational style for specifying analyses across programming paradigms, introduced by Nielson and Nielson [NN97, NN98, NN02], and with Hankin [NNH99]. By abstracting from domain specific formalisms and instead using standard mathematical notations, the Flow Logic constitutes a meta-language that can present an analysis without requiring additional knowledge about particular formalisms. Deriving an analysis estimate from the resulting analysis specification is then left as a separate activity, usually involving orthogonal considerations and tools. This approach allows the designer to focus on the specification of analyses without making compromises dictated by implementation considerations. Similarly, implementation is simplified and improved, as the implementer is always free to choose the best available tool. In the next sections, we will present control flow analysis of LySA in the style of flow logic.

The control flow analysis that we use in protocol analysis is specified using the flow logic framework as a predicate

$$\rho, \kappa, \psi \models P$$

that holds precisely when ρ, κ, and ψ form an analysis result that correctly describes the behaviour of the process P.

The main components of the analysis are:

- *The variable environment ρ,* an over-approximation of the potential values of each variable that it may be bound to.
- *The network component κ,* an over-approximation of the set of messages that can be communicated over the network.

Table 3.6: Analysis for Terms, $\rho \models E{:}\vartheta$

(AName)	$\dfrac{\lfloor n \rfloor \in \vartheta}{\rho \models n{:}\vartheta}$
(AVar)	$\dfrac{\rho(\lfloor x \rfloor) \subseteq \vartheta}{\rho \models x{:}\vartheta}$
(AEnc)	$\dfrac{\begin{array}{c}\wedge_{i=0}^{k} \rho \models E_i{:}\vartheta_i \quad \wedge \\ \forall V_0, V_1, \ldots, V_k{:} \wedge_{i=0}^{k} V_i \in \vartheta_i \quad \Rightarrow \quad \{V_1, \ldots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta\end{array}}{\rho \models \{E_1, \ldots, E_k\}_{E_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,]{:}\vartheta}$

- *The error component* ψ, the set of error messages in the form (ℓ, ℓ'), indicating that something encrypted at ℓ was unexpectedly decrypted at ℓ'.

The analysis is judgments of the form $\rho, \kappa, \psi \models P$ which express that ρ, κ, ψ compose a valid analysis for the process P. We also need to introduce the auxiliary judgment $\rho \models E{:}\vartheta$ at this point. This expresses that ϑ, the set of values, is an acceptable estimate of the values that the term E may evaluate in ρ, the abstract environment.

To keep the analysis component finite, we partition all the names that are generated by a LySA process into finitely many equivalence classes. A *canonical value* is a representative for each of these equivalence classes. Names from the same equivalence class are assigned a common *canonical name* and instead of the actual names, we use the names of those equivalence classes. For example, the canonical representative of a name n is denoted by $\lfloor n \rfloor$. Since it allows us to analyse an infinite number of principals, canonical value is an important analysis element [BRNN04].

The analysis of terms is listed in Table 3.6. The rule for analysing names (AName) states that ϑ is an acceptable estimate for a name n if the canonical representative of n belongs to ϑ. The rule for analysing variables (AVar) states that ϑ is an acceptable estimate for a variable x if it is a superset of $\rho(\lfloor x \rfloor)$. The rule for analysing symmetric encryption (AEnc) finds the set ϑ_i for each term E_i, collects all k-tuples of values $(V_0, \ldots, V_k)$ taken from $\vartheta_0 \times \ldots \times \vartheta_k$ into values of the form $\{V_1, \ldots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,]$ and requires that these values belong to ϑ.

The analysis of processes is listed in Table 3.7. The idea of the analysis is very similar to the analysis of terms, therefore instead of explaining all the rules we explain only one interesting rule. The rule for analysing output (AOut) uses the

Table 3.7: Analysis for Processes, $(\rho,\kappa) \models P{:}\psi$

(ANil)	$(\rho,\kappa) \models 0{:}\psi$
(APar)	$\dfrac{(\rho,\kappa) \models P_1{:}\psi \quad \wedge \quad (\rho,\kappa) \models P_2{:}\psi}{(\rho,\kappa) \models P_1 \mid P_2{:}\psi}$
(ARep)	$\dfrac{(\rho,\kappa) \models P{:}\psi}{(\rho,\kappa) \models !P{:}\psi}$
(ANew)	$\dfrac{(\rho,\kappa) \models P{:}\psi}{(\rho,\kappa) \models (\nu\, n)\, P{:}\psi}$
(AOut)	$\dfrac{\begin{array}{c}\wedge_{i=1}^{k}\rho \models E_i{:}\vartheta_i \quad \wedge \\ (\rho,\kappa) \models P{:}\psi \quad \wedge \\ \forall V_1,\ldots,V_k{:}\, \wedge_{i=1}^{k} V_i \in \vartheta_i \quad \Rightarrow \quad \langle V_1,\ldots,V_k\rangle \in \kappa\end{array}}{(\rho,\kappa) \models \langle E_1,\ldots,E_k\rangle.P{:}\psi}$
(AIn)	$\dfrac{\begin{array}{c}\wedge_{j=1}^{k}\rho \models E_i{:}\vartheta_i \quad \wedge \\ (\rho,\kappa) \models P{:}\psi \quad \wedge \\ \forall V_1,\ldots,V_k \in \kappa{:}\, \wedge_{j=1}^{k} V_i \in \vartheta_i \quad \Rightarrow \quad \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor)\end{array}}{(\rho,\kappa) \models (E_1,\ldots,E_j;\, x_{j+1},\ldots,x_k).P{:}\psi}$
(ADec)	$\dfrac{\begin{array}{c}\rho \models E{:}\vartheta \quad \wedge \\ \forall\, \wedge_{i=0}^{j}\, \rho \models E_i{:}\vartheta_i \quad \wedge \\ (\rho,\kappa) \models P{:}\psi \wedge \\ ((\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L}) \Rightarrow (\ell,\ell') \in \psi) \quad \wedge \\ \forall \{V_1,\ldots,V_k\}_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta{:}\, \wedge_{i=0}^{j} V_i \in \vartheta_i \quad \Rightarrow \quad \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor)\end{array}}{(\rho,\kappa) \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1,\ldots,E_j;\, x_{j+1},\ldots,x_k\}_{E_0}^{\ell'}[\mathsf{orig}\,\mathcal{L}\,]\ \mathsf{in}\ P{:}\psi}$

analysis for terms to find the estimate ϑ_i for each term E_i and requires that all all k-tuples of values $\langle V_1,\ldots,V_k\rangle$ taken from $\vartheta_1 \times \ldots \times \vartheta_k$ are in κ (i.e. they may flow on the network). The rule also requires that the components ρ,κ,ψ compose a valid analysis for process P.

Example 3.5 (Static Analysis Results) *Static analysis of the* LySA *model given in Example 3.4 will lead to the following results:*

$\langle A, B, K_A, \{K\}^{\ell_A}_{K_A}[\mathsf{dest}\,\ell_{\mathcal{B}}\,]\rangle \in \kappa$
$K_A \in \rho(x_{KA})$
$\{K\}^{\ell_A}_{K_A}[\mathsf{dest}\,\ell_{\mathcal{B}}\,] \in \rho(x)$
$K \in \rho(x_K)$

Looking at the results above, it is easy to see that the first line is related to ***line a*** *in Example 3.4. Likewise, next two lines derived from* ***line b*** *and the last line derived from* ***line c*** *in Example 3.4. Note that, the details on how the analysis works and thus how Example 3.4 leads to Example 3.5 is present in [BBD+05].*

■

3.3.2 Attacker Model

In practice, network protocols are vulnerable to attacks. Unfortunately it is even easier to attack wireless networks since any computer within range that is equipped with a wireless client card can pull the signal and access the data. In this study, LySA processes are analysed in parallel with the Dolev-Yao attacker [DY81]. The operations that this attacker model can perform are listed below, but before this we have to introduce new canonical (see Section 3.3.1) names and variables for the attacker. All the canonical names of the attacker are mapped to $n_\bullet$ and all the canonical variables of the attacker are mapped to $z_\bullet$. We also have $\ell_\bullet$ which is a crypto-point in the attacker.

The descriptions of the Dolev-Yao conditions are:

- The attacker initially has the knowledge of the canonical name $n_\bullet$ and all free names of the process P but he can improve his knowledge by eavesdropping on all messages sent on the network.
- The attacker can improve his knowledge by decrypting messages with the keys he already knows. Unless the intended recipient of the message was an attacker, an error (ℓ,$\ell_\bullet$) should be added to the error component ψ which means that something encrypted at ℓ was actually decrypted by the attacker at $\ell_\bullet$.
- The attacker can construct new encryptions using the keys he already knows. If this message is received and decrypted by a principal, then an error ($\ell_\bullet$,ℓ) should be added to the error component ψ which means that

something encrypted at the attacker was decrypted by the attacker by a process P at ℓ.

- The attacker can send messages on the network using his knowledge and thus forge new communications.

These conditions enable the attacker to establish scenarios including eavesdropping, modification, man-in-the-middle and replay attacks. The soundness of the Dolev-Yao condition is proved in [BBD$^+$05].

As shown in Fig. 3.1, the LySA model of a protocol is analysed in parallel with the attacker model and processed by the LySA-tool (see Section 3.4.4) which implements the static analysis [LyS]. The results of the analysis are used to validate destination/origin authentication and confidentiality properties of the protocols. If no violation is detected, namely the error component ψ is empty, then it is guaranteed that the protocol satisfies the destination/origin authentication properties. Furthermore, the potential values that are learned by the attacker help us in validating the confidentiality properties. The details as well as the proof of the soundness of the analysis are presented in [BBD$^+$03].

Example 3.6 (Presence of Attackers) *In Example 3.5, we analysed Example 3.4 in an attack-free setting. Now we add the attacker model and get the following results in addition to the results in Example 3.5. Since the attacker is able to learn everything sent on the network we have:*

$$K_A, \{K\}_{K_A}^{\ell_A}[\mathsf{dest}\,\ell_{\mathcal{B}}\,] \in \rho(z_\bullet)$$

Therefore, the attacker can decrypt the encrypted part of the message which leads to the violation:

$$(\ell_A, \ell_\bullet) \in \psi$$

Thus we conclude that the encryption at crypto-point ℓ_A which was intended to be decrypted at ℓ_B can be decrypted by the attacker and hence the example protocol is flawed. ■

3.4 Application on ZigBee Wireless Sensor networks

In this section, we present an application of the static program analysis method that we explained in this chapter. This application has many features that make it interesting. First of all, it pinpoints an undiscovered and non-trivial flaw in a real cryptographic security protocol. Another key issue is that the protocol is being used in one of the latest wireless sensor network standards, ZigBee, that is promising and emerging in the sensor networks field. Therefore, the protocol includes components that are known to be secure when they are individually used and some of them are industry standards such as SKKE (see details in Chapter 2, Section 2.1.5.2). Still we show that combining proven to be secure components is not sufficient for guaranteeing security properties. Last feature of this application is that we use protocol analysis not only to discover flaws but also to verify our fixing proposals.

3.4.1 ZigBee-2007 End-to-End Application Key Establishment Protocol

As we explained in Chapter 2, ZigBee is a fairly new but promising WPAN standard for wireless sensor networks that have very low resource requirements. In parallel with this, the devices that operate in ZigBee networks have limited resources in terms of memory, processor, storage, power, etc. Therefore implementing the security guarantees is a great challenge and the verification of the security properties is of paramount importance.

We start by reminding some of the key points that are necessary for a clear understanding of the development, and we omit all the details which are not directly relevant to this study. However, in addition to Chapter 2 a detailed survey on ZigBee security can be suggested as [YNN08] and surely the ultimate source is the ZigBee documentation [Zig08d, Zig08e, Zig08b, Zig08a, Zig08c] which is a rather difficult read with hundreds of pages including references to several other standards.

End-to-End Application Key Establishment is the protocol to be used when establishing a Link Key (LK) between two ZigBee devices, which are running in *High Security Mode* (which was called *Commercial Mode* in the previous standard, ZigBee-2006 [Zig06]). We will call the devices as initiator (**A**) and responder (**B**). Note that there is also a Trust Center (**TC**), which shares a pairwise secret key with each principal in the network. TC is actually an ap-

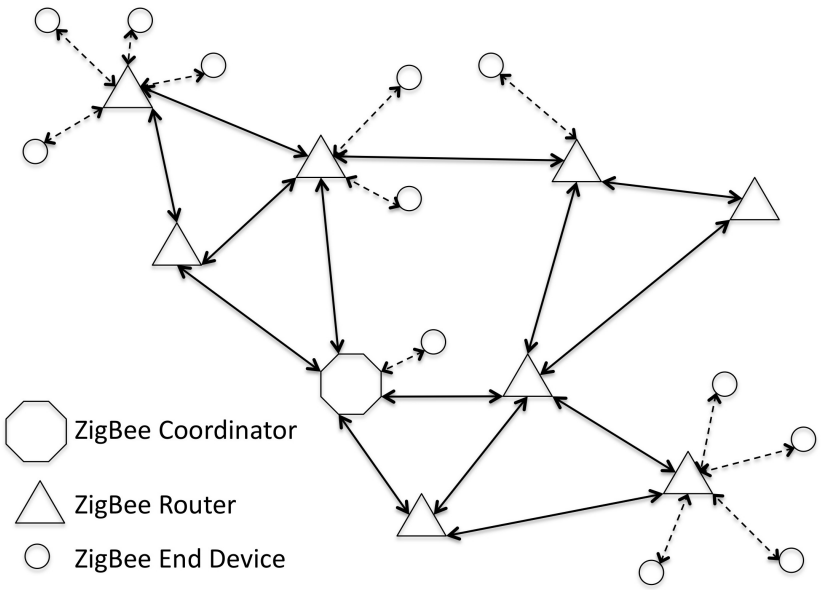


Figure 3.3: ZigBee Network Model

plication that runs on a preferably more powerful ZigBee device referred to as *ZigBee Coordinator* which is unique in the network; whereas the remaining devices might be of type *ZigBee Router* or *ZigBee End Device*, as shown in Fig. 3.3. For a better understanding we should mention that for *two* ZigBee devices to establish a (pairwise) secure communication, they must share a symmetric key (LK) which they either *receive* from a trusted server (TC) or *create mutually* using a temporary key received from the trusted server.

ZigBee-2007 End-to-End Application Key Establishment Protocol has two different cases according to the configuration of TC, we will call them as *Case 1* and *Case 2*. In Case 1, TC creates the LK itself and sends it to each principal. Therefore, the initiator and the responder have no role in the creation of the LK. In Case 2, TC creates a temporary shared key called *Master Key* (**MK**) and sends it to each principal. Using this MK, A and B initiate a Symmetric-Key Key Establishment (**SKKE**) procedure to establish an LK. This case allows principals to create an LK *mutually*. SKKE is actually a key agreement scheme employed in the ZigBee End-to-End Application Key Establishment mechanism, and its components are defined in the ANSI X9.63-2001 standard [Ame01]. We have explained the necessary details of the SKKE protocol in the previous chapter. At the end of (a successful run of) either case, two ZigBee devices will be able to establish secure communication using their pairwise encryption key, LK.

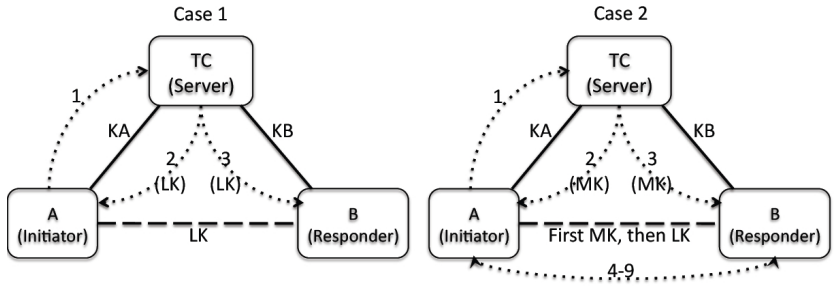


Figure 3.4: ZigBee-2007 End-to-End Application Key Establishment Protocol

The two cases of the End-to-End Application Key Establishment protocol are visualized in Fig. 3.4. The solid lines represent the already secure communication paths, labeled by corresponding symmetric encryption keys. The dashed lines represent the resulting secure communication paths after a successful protocol run, again labeled by corresponding encryption keys. Finally, the dotted lines are the messages in the protocol labeled by their sequence numbers and the encryption keys they deliver.

3.4.1.1 Case 1: *Nonmutual Key Establishment*

In Case 1, the initiator begins the procedure of establishing an LK with the responder by sending TC the first message, **request key**, which includes *destination address* (=TC), *requested key type* (=Application Key), and *partner address* (=B). Then TC creates an LK for two principals, and sends it to each principal in two similar **transport key** messages. Since TC is configured to send an LK directly in this case, the *key type* value in the last two messages will be Application Link Key (AppLK). The only difference between these two messages is a boolean value that indicates the initiator (TRUE: message recipient is the initiator, FALSE: message recipient is the responder), and also the principal address'. All the messages in this case are encrypted with the sender/receiver principal's key that is shared with TC (assuming that the security suite is *Encryption-only*). The type of this key can either be Trust Center Link Key (**TCLK**) or Trust Center Master Key (**TCMK**), as defined in the ZigBee specification [Zig08d], but for simplicity we will call it *KA* for principal A, and *KB* for principal B. The protocol narration of Case 1 is given in Table 3.8.

As a side note, we have used a simplified version of this protocol previously in this chapter to demonstrate how a protocol narration is converted into first an extended protocol narration, then a LySA model in Table 3.3, Table 3.4, and

Table 3.8: Protocol Narration - Case 1

1.	**A→TC:** {TC, AppKey, B}$_{KA}$
2.	**TC→A:** {A, AppLK, B, TRUE, LK}$_{KA}$
3.	**TC→B:** {B, AppLK, A, FALSE, LK}$_{KB}$

Table 3.9: Protocol Narration - Case 2

1.	**A→TC:** {TC, AppKey, B}$_{KA}$
2.	**TC→A:** {A, AppMK, B, TRUE, MK}$_{KA}$
3.	**TC→B:** {B, AppMK, A, FALSE, MK}$_{KB}$
4.	**A→B:** {B, FALSE, Zero, SKKE}$_{MK}$
5.	**B→A:** {A, TRUE}$_{MK}$
6.	**A→B:** {NA}$_{MK}$
7.	**B→A:** {NB}$_{MK}$
8.	**A→B:** MAC{3,A,B,NA,NB}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
9.	**B→A:** MAC{2,B,A,NB,NA}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$

Table 3.5, respectively.

3.4.1.2 Case 2: *Mutual Key Establishment*

In Case 2, the first three messages are almost the same as in Case 1, except in this case TC is configured to send MK, and therefore key type is the Application Master Key (AppMK). The rest of the messages are between the initiator and the responder. In the fourth message, **establish key**, A sends B his request to start SKKE. The values, False and Zero, indicate that there is no *parent* (router, TC, etc.), and no *parent address*, respectively. The fifth message is the response of B to A's SKKE request. Note that these two messages are encrypted by MK, which was received in the previous two messages. The remaining four messages are actually the SKKE protocol itself. Messages 6 and 7 include the *challenges* (NA, NB) of the principals. Messages 8 and 9 are the complex messages which can be computed by both parties to verify each other. A and B create two *message authentication codes* (**MAC**) using their knowledge, besides the MAC key itself is a *hash* (H) of another MAC which they produce using the same knowledge [Fed02]. After the verification, the new LK will be H(MAC{A,B,NA,NB}$_{MK}$, 2), which is a minor variation of the MAC key that was used in the last two messages. The protocol narration of Case 2 is given in Table 3.9.

3.4.2 The Flaw

In wireless networks, it is easy to intercept, forge and inject messages. Without any formal analysis, an experienced eye can see that all the messages in ZigBee-2007 End-to-End Application Key Establishment Protocol can be replayed when the same long-term encryption keys (KA, KB) are still being used. The reason is the lack of *freshness* elements like nonces, timestamps, etc. This flaw can lead to serious replay attacks, denial of service (DoS) attacks, etc. Even worse, when an old session key is compromised, an attacker can decrypt all the messages by replaying that old session key. In other words, lack of freshness can cause failures in *authenticity* (in the case that principals accept an old session key from a rogue TC) and *confidentiality* (in the case that principals start using a compromised session key).

As can be seen in the narration of the protocol, no freshness indicator is used in the distribution of either LK (in Case 1) or MK (in Case 2, the first three messages). Therefore, all the messages can be replayed. Replay of a message that includes a key is very critical. An attacker can store a message including a key from a previous run of this protocol, and then send the old message to make principals communicate using this old key. If the old key is compromised, then the attacker will be able to decrypt all the messages between two victim principals.

The significance of the security risk that is caused by this flaw may require more explanation. Indeed, the flaw does not disclose any session key but allow reuse of a former key. Besides, brute force attacks or other types of known cryptographic attacks for obtaining the key do not seem practical for the current specification (i.e. the keys are 128-bits). However, disclosure of a key might still be possible without dealing with cryptography, and reuse of an old session key can cause serious risks. An example scenario is given below:

Scenario 1 *A and B established a link key, and had secure communication with the help of that pairwise key. Than B left the network and disclosed the key, which might be by means of hardware (e.g. local key extraction from the chipset such as connecting a debugger, erasing the chip, then freely reading the contents of RAM), or software (e.g. a bug in the implementation that discloses the key after the session expires or terminates with the natural assumption that a new session key will be used for a future session) defects. If B rejoins the network, and run the key establishment protocol with A (no matter which case or security level is chosen), the disclosed key may be replayed by the attacker who can decrypt all the communication using the disclosed key.*

In the ZigBee Specification, the notion of *frame counter* is emphasized as the

Table 3.10: Attack Scenario - Case 1

1. A→TC: {TC, AppKey, B}$_{KA}$
2. TC→A: {A, AppLK, B, TRUE, LK}$_{KA}$
3. TC→B: {B, AppLK, A, FALSE, LK}$_{KB}$
1'. A→TC: {TC, AppKey, B}$_{KA}$
2'. M(TC)→A: {A, AppLK, B, TRUE, LK}$_{KA}$
3'. M(TC)→B: {B, AppLK, A, FALSE, LK}$_{KB}$

freshness protection. This approach is not a strong one for several reasons. First of all, a frame counter uses incrementing values rather than random values and rejects frames with a smaller counter value. Second, regardless of the length (which is 32-bits in ZigBee) it is easy to cause overflow to frame counters. As indicated in [SW04], if an adversary forges a frame with the maximum value (*i.e. 0xFFFFFFFF*) any further frame will be rejected. In addition, using counters is not a novel approach, since in such layered architectures lower layers also used similar counters.

3.4.2.1 Flaw in Case 1

The attack scenario for Case 1 is given in Table 3.10. The first run (messages 1 to 3), is an old run which is intercepted by an attacker. Here, it is appropriate to mention that LK is used like a session key and KA/KB are used like master keys. Therefore, KA and KB are possibly the same in two different runs. The second run in the attack scenario (messages 1' to 3') is initiated regularly, but the last two messages are replayed by the attacker using the messages that are captured from the old run. Furthermore, the attacker does not necessarily need to wait for a message like 1' since he can already replay it, too.

3.4.2.2 Flaw in Case 2

The attack for Case 1 is also possible for Case 2, in which MK is sent without any freshness indicator. Even though LK is created mutually by the use of SKKE in Case 2, a compromised old MK that is replayed to principals before SKKE will allow an attacker to create the LK as well. The attack scenario for Case 2 is given in Table 3.11. The first run (messages 1 to 9) is the old run and it is sufficient for an attacker to capture messages 2 and 3. Then the attacker replays these messages in the new run (messages 1' to 9'). Although the nonce's used in SKKE (exchanged in messages 6 and 7) are different, as long as MK is compromised the attacker can decrypt these messages and learn the nonces as

Table 3.11: Attack Scenario - Case 2

1. A→TC: {TC, AppKey, B}$_{KA}$
2. TC→A: {A, AppMK, B, TRUE, MK}$_{KA}$
3. TC→B: {B, AppMK, A, FALSE, MK}$_{KB}$
4. A→B: {B, FALSE, Zero, SKKE}$_{MK}$
5. B→A: {A, TRUE}$_{MK}$
6. A→B: {NA}$_{MK}$
7. B→A: {NB}$_{MK}$
8. A→B: MAC{3,A,B,NA,NB}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
9. B→A: MAC{2,B,A,NB,NA}$_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
1'. A→TC: {TC, AppKey, B}$_{KA}$
2'. M(TC)→A: {A, AppMK, B, TRUE, MK}$_{KA}$
3'. M(TC)→B: {B, AppMK, A, FALSE, MK}$_{KB}$
4'. A→B: {B, FALSE, Zero, SKKE}$_{MK}$
5'. B→A: {A, TRUE}$_{MK}$
6'. A→B: {NA'}$_{MK}$
7'. B→A: {NB'}$_{MK}$
8'.A→B:MAC{3,A,B,NA',NB'}$_{H(MAC\{A,B,NA',NB'\}_{MK},1)}$
9'.B→A:MAC{2,B,A,NB',NA'}$_{H(MAC\{A,B,NA',NB'\}_{MK},1)}$

well. As a result, the attacker can still compute the new LK which is actually H(MAC{A,B,NA',NB'}$_{MK}$, 2) (see Section 3.4.1). Therefore, we may conclude that the flaw is critical in both cases.

3.4.3 Proposed Fixed Protocols

We propose fixed protocols that use nonces to ensure freshness of the messages and at the same time the keys. We make use of the vital principles defined on [AN94]. The narrations of our proposed solution are given in Table 3.12 and Table 3.13 for Case 1 and Case 2, respectively.

In Case 1, we added the nonce of the initiator (**NA**) to the first two messages. This will ensure that when receiving the second message, A will believe that she is communicating with the TC who knows her nonce and also her private key. Note that message 1 can still be replayed but it will be ignored if A does not verify message 2. We inserted two more messages before the last message, so that we use nonces of the TC (**NTC**) and the responder (**NB**) to avoid replay attacks. This will ensure that when receiving the fifth message, B will believe that he is communicating with TC who knows his nonce. Also note that message 3 can still be replayed but the process will be ignored if B does not

Table 3.12: Proposed Fix - Case 1

1.	**A→TC:** $\{$TC, AppKey, B, NA$\}_{KA}$
2.	**TC→A:** $\{$A, AppLK, B, TRUE, NA, LK$\}_{KA}$
3.	**TC→B:** $\{$B, A, NTC$\}_{KB}$
4.	**B→TC:** $\{$TC, A, NTC, NB$\}_{KB}$
5.	**TC→B:** $\{$B, AppLK, A, FALSE, NB, LK$\}_{KB}$

Table 3.13: Proposed Fix - Case 2

1.	**A→TC:** $\{$TC, AppKey, B, preNA$\}_{KA}$
2.	**TC→A:** $\{$A, AppMK, B, TRUE, preNA, MK$\}_{KA}$
3.	**TC→B:** $\{$B, A, NTC$\}_{KB}$
4.	**B→TC:** $\{$TC, A, NTC, preNB$\}_{KB}$
5.	**TC→B:** $\{$B, AppMK, A, FALSE, preNB, MK$\}_{KB}$
6.	**A→B:** $\{$B, FALSE, Zero, SKKE$\}_{MK}$
7.	**B→A:** $\{$A, TRUE$\}_{MK}$
8.	**A→B:** $\{$NA$\}_{MK}$
9.	**B→A:** $\{$NB$\}_{MK}$
10.	**A→B:** MAC$\{$3,A,B,NA,NB$\}_{H(MAC\{A,B,NA,NB\}_{MK},1)}$
11.	**B→A:** MAC$\{$2,B,A,NB,NA$\}_{H(MAC\{A,B,NA,NB\}_{MK},1)}$

verify message 5.

Our solution is also applicable to the leaked MK problem in Case 2. Similar to our solution for Case 1, we change the first three messages of Case 2 with five messages that are also given in Table 3.13. Not to confuse with the nonces used in SKKE, the nonces we added are called (**preNA**) and (**preNB**) in Case 2.

The fix that we propose is a mechanism that suffices to fix the flaws in the original protocol. There might be other ways to fix, but this is a solution that simply works and has proven (by formal verification) to be secure.

Obviously, the proposed solution would come at a particular cost. Particularly, the number of messages in each protocol is increased by two, and the usage of nonces are required. Transmission of more messages means more power consumption, but for security critical applications (e.g. in Smart Energy, Commercial Building Automation, etc.) this kind of fix which ensures that TC is authenticated to both A and B (i.e. the new LK is not replayed) is necessary, so the additional messages are inevitable. Besides the original protocol in Case 2 already has nine messages (whereas the primitive version, Case 1, only has three), which is a proof that in order to have a sound protocol ZigBee may have longer protocols for the same purpose. The usage of nonces is not a new cost

since it is already in SKKE which is employed by Case 2. However, the freshness is preserved for only SKKE but not the protocol itself due to the design mistake of the wrapping protocol.

As we mentioned before, the flaw in End-to-End Application Key Establishment protocol may be visible to an experienced eye but to claim that a fix is flawless, verification using formal methods is crucial. *Static analysis with* LySA is one of the methods that can be used, which has many advantages such as scalability and the guarantee of termination.

3.4.4 Formal Verification Details

Analysing security protocols without any formal verification method is not a reliable way to find flaws, nor to guarantee that there are no flaws. To make our assertions and arguments sound, we use static analysis to analyse protocols. To be finite, this method is computing over-approximations rather than exact answers, and therefore may lead to false positives. However, when the analysis results tell that the protocol is error-free, then it really is. In other words, no simulation or verification is necessary when the protocols successfully passes static analysis.

The base protocols in Section 3.4.1 are modelled using LySA process calculus and analysed using the LySA-tool [LyS]. The result supports our claims in Section 3.4.2. The base protocols are prone to replay attacks which will cause serious problems in the case of a leaking key.

The proposed protocols in Section 3.4.3 are also modelled analysed in the same way with the base protocol. The result is successful, namely the proposed protocols do not have any flaws.

The settings that we use to implement the LySA model and verify in the LySA-tool are listed below:

- we check for the origin and destination addresses in each message (by adding them as prefixes such as in IPv4 or IPv6)
- we have the necessary annotations for the encryptions and decryptions
- we allow legitimate attackers in addition to the illegitimate attackers (by adding appropriate zero indices, namely attacker also shares master key with TC)

- we model three groups of (infinite) principals so that we can model man-in-the-middle attacks
- we add an extra message that is encrypted using the session key (to see whether the compromised key can be used)
- we check all the fields in the messages to have proper values (by pattern matching), except session keys which are newly created (and bound to variables in inputs)

To distinguish between old rounds and new rounds of the protocol we apply a new technique in LySA. We add round indicators to the end of pattern-matched fields in messages and match them in a smart way to distinguish old runs. Using this technique, we can investigate replay attacks successfully.

3.5 Discussion

Analysing protocols is not a trivial issue, and in this chapter we presented an analysis method with a detailed application on a new and so called enhanced security protocol that uses secure components.

In this approach, we have solid benefits in mainly:

- *solutions always exist and are computed in polynomial time.* This is an important advantage because approaches based on model checking cannot always guarantee termination, and besides prone to state space explosion problem. Besides the analysis is correct with respect to formal operational semantics, which may be hard to establish in different approaches such as the ones based on modal logic of beliefs (BAN) where the completeness property does not generally hold.

However, those benefits come with a particular cost:

- *lack of trace and counter-example.* Due to the nature of the analysis, there is no trace and no produced counter-example to help flaw discovery. As a result of the over-approximation, false positives may occur and manual inspection is required to match the reported violations to actual flaws.

Another thing we have presented was the usage of protocol analysis in suggesting a secured version of a flawed protocol. Fixing the flaws and proposing secure

protocols is another non-trivial job. In this manner, we made use of prudent engineering practices of Gordon and Abadi [AN94], and benefited fruitful discussions with Gavin Lowe. One of the points we emphasized was the importance of freshness, and the importance of proper usage of freshness indicators such as nonces, challenges, etc.

We can recapitulate as encryption is not synonymous with security, and its improper use can lead to errors. The proper use should be verified by protocol analysis methods that focus on certain security properties.

Part II

Quantitative Analysis

Chapter 4

Preliminaries for Stochastic Model Checking

Stochastic Model Checking is a powerful method that is used for computing the likelihood of the occurrence of certain events in a system. As in conventional model checking, a description of a model together with a specification is taken as inputs, then it is checked whether or not the model satisfies the specification. In addition to this, in stochastic model checking calculation of actual probabilities is involved via numerical and analytical methods [KNP07].

In this chapter, we introduce the necessary background material for the following chapters of this thesis. Although the topic is rather extensive, we chose a subset that we will use in the following chapters.

The chapter is organized as follows: In Section 4.1, we introduce the concept of model checking and how it evolved. In Section 4.2, we introduce how systems can be modelled in order to be suitable for model checking. Then in Section 4.3, we focus on how to model the properties that we would like to verify using model checking. Following is Section 4.4, where we mention the algorithms and explain the model checking implementation that we will use in the following chapters.

4.1 An Overview of Model Checking

Model checking is an eminent formal verification technique for assessing properties of systems. This technique provides algorithmic means of determining whether an abstract model of a system satisfies a formal specification expressed as a temporal logic formula. In case of violation of a property, model checking also identifies a counterexample execution that shows the problematic point.

As the name suggests, *model* of a system should be developed, and desired properties should be specified for model checking. Typically, the systems under consideration are hardware or software systems, and properties to be checked are absence of deadlocks, invariants, request-response and timing properties, etc. Model checking systematically checks if a given model satisfies specified properties, by exploring all possible system states in a brute-force manner.

In early 1980s, Edmund M. Clarke and E. Allen Emerson from USA, and Jean Pierre Quielle and Joseph Sifakis from France, independently authored seminal papers proposing essentially the same method, what has become the highly successful field of model checking [CJ97], [QS82]. Through years, the field of model checking has evolved and finally the method was awarded the 1998 ACM Paris Kanellakis Award for Theory and Practice, and its founders were awarded the 2007 Turing Award given by the Association for Computing Machinery.

Model checking can be described as a process involving three phases [BK08]. First phase is the *modelling* phase, where the system under consideration is modelled using a model description language and the property under consideration is formalized using a property specification language. Then comes the *running* or *checking* phase, where a model checker runs for checking the validity of the property in the model of the system. This is where an intelligent exhaustive search of the state space is done to determine if the specification is true or not. Last phase is the *analysis* phase, where a counterexample can be generated if the property is violated, and the design will be revised to fix the flaw. Then, model checking process will be restarted with a refined model until the property is satisfied.

At this point, we need to clarify the distinction between *verification* and *validation* using the definitions from [BK08]. Validation is the judgement of whether the formalized problem statement, i.e. the model and the properties, is an adequate description of the verification problem. Thus, it helps us checking that we are verifying the right thing. On the other hand, verification is the process of checking that the design satisfies identified requirements, as we explained above.

One of the main advantages in using model checking is, it provides useful diag-

nostic information when a property is validated, i.e. relevant counterexample is generated. Another advantage is, model checking provides a *push-button*, i.e. automated, method for verification. Generally speaking, both bug detection, as well as correctness verification is offered.

The major problem in model checking is the state-space explosion, that is number of global states in a concurrent system getting enormous. Another weak point is that since model checking verifies the abstract model of a system, the results will be incorrect if the model is not faithful to to the original system.

Examples of the use of model checking include verification of hardware circuits, communication protocols, software device drivers, real-time embedded systems, and security algorithms, etc.

4.2 Modelling

A *model* is the main input to model checking, where the behaviour of the system under consideration is described in an accurate and unambiguous manner. In general, the concept of finite-state machine that consists of a finite set of states and a set of transitions is employed. States of a model stores information about the current values of the variables, and transitions between states describe how the system evolves from one state into another.

The importance of the *model* in model checking is elegantly summarized by [BK08] as:

> Any verification using model-based techniques is only as good as the model of the system.

In the rest of this section, we will start from the simplest mathematical models and proceed to the models that we use in this study.

4.2.1 Labelled Transition Systems

A simple way of characterising systems is by using the notion of *labelled transition systems*. A labelled transition system $LTS = (S, s_{init}, \longrightarrow, Act, AP, L)$, consists of:

- a set of states S
- an initial state $s_{init} \in S$
- a labelled transition relation $\longrightarrow \subseteq S \times Act \times S$
- a set of actions Act
- a set of atomic propositions AP, and
- a labeling function $L : S \longrightarrow 2^{AP}$

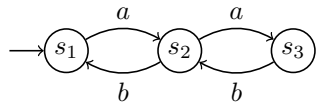
Traditionally and for convenience, we write $s \xrightarrow{\alpha} t$ to mean that $(s, \alpha, t) \in \longrightarrow$.

■

Example 4.1 (Labelled Transition System) *The labelled transition system*

$$LTS = (\{s_1, s_2, s_3\}, s_1, \{(s_1, a, s_2), (s_2, a, s_3), (s_3, b, s_2), (s_2, b, s_1)\}, \{a, b\}, AP, L)$$

abstractly describes a system with three states, two labels, and four distinct labelled transitions. The state-transition diagram of the system can be drawn as follows:



Here we can imagine that this is a very simple network model and define first atomic propositions as $AP = \{one\_device, two\_devices\}$, then labeling function as $L(s_1) = \emptyset), L(s_2) = \{one\_device\}, L(s_3) = \{two\_devices\}$. By doing that, we labelled the states with meaningful labels. We can also replace the action a with join, and action b with leave to represent the actions of the system neatly. ■

4.2.2 Markov Chains

Up to now, we have introduced modelling for traditional model checking, that is forming labelled transition systems. However, in the probabilistic setting we use models which also incorporate information about the likelihood of transitions occurring. In this section, we introduce Markov chains as the formalism we will employ in the probabilistic settings. We will focus on Discrete-Time Markov Chains (**DTMC**) and Continuous-Time Markov Chains (**CTMC**) as necessary for the following chapters.

Markov chains are the most suitable operational model for the evaluation of performance and dependability. In the simplest definition, Markov chains are transition systems that have probability distributions for the successor states. In other words, being in a certain state, the next state is chosen probabilistically rather than nondeterministically. Note that nondeterministic choice can also be included in models such as Markov Decision Processes (MDP) or Continuous-Time Markov Decision Processes (CTMDP), but not in Markov chains.

The major property of Markov chains, is the system evolution does not depend on the history. The probability distribution to choose the next state only depends on the current state. This property is known to be the *memoryless property*.

Below we introduce discrete-time and continuous-time Markov chains in turn. Note that we explain DTMC even though we only use CTMC in this thesis. The reason for including DTMC is that in certain cases (such as computing transient probabilities) CTMCs are uniformised to DTMCs to allow efficient model checking.

4.2.2.1 Discrete-Time Markov Chains

A *discrete-time Markov chain* (DTMC) $\mathcal{M} = (S, s_{init}, \mathbf{P}, AP, L)$, consists of:

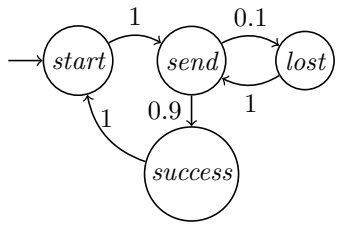
- a countable, non-empty set of states S
- an initial state $s_{init} \in S$
- a transition probability matrix $\mathbf{P} : S \times S \longrightarrow [0, 1]$ such that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$
- a set of atomic propositions AP, and
- a labeling function $L : S \longrightarrow 2^{AP}$

■

In a DTMC, the system changes state *at each time step* in accordance with the transition probability matrix $\mathbf{P}$. The transition probability matrix is an $n \times n$ matrix $\mathbf{P}$, n being the number of the states. The row $\mathbf{P}(s, \cdot)$ specifies the probabilities of moving from state s to its successor states. This matrix specifies, for each state s, the probability $\mathbf{P}(s, s')$ of moving from s to s' by a single transition. Obviously, $\mathbf{P}(s, s') \in [0, 1]$, in addition we further demand that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ in order for the system to be well-formed.

In this situation it is meaningful to choose the set of labels to be the set of permissible probabilities, i.e. $p \in [0,1]$, and the resulting notion of labelled transition system is exactly the class of *discrete-time Markov chains*.

Example 4.2 (Discrete-time Markov chain) *Consider a simple communication channel such that not all the messages sent can reach the destination. Model of the channel is given as a DTMC below:*



A message is generated in the start state, and sent in the sent state. With probability of 0.9 *the transmission will be successful, however with probability of* 0.1 *the message will be lost. In the latter case, the message will be sent again until it is successfully received by the destination. After the message gets delivered, the system will return to the start state.*

The transition probability matrix $\mathbf{P}$ *of this example is:*

$$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0.1 & 0.9 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

■

For the sake of simplicity, we often specify DTMCs as a tuple of $(S, \mathbf{P}, L)$.

A *path* through a DTMC is a sequence of states $\sigma = s_0 s_1 s_2 \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. The expression $\sigma[i]$ is used to denote the $(i+1)$th state of σ, i. e., $\sigma[i] = s_i$. We write $Paths(s)$ to be the set of paths such that $\sigma[0] = s$. ■

4.2.2.2 Continuous-Time Markov Chains

The main type of model that we use for model checking throughout this thesis, *continuous-time Markov chain* (CTMC), extends DTMC models. In DTMCs,

each transition corresponds to a discrete-time step, while in CTMCs transitions can occur in real time. Each transition is labelled with a *rate*, which defines the delay that occurs before the transition is taken. The delays are sampled from an exponential distribution that uses this rate as a parameter.

Using our simplified definition of DTMC, we define A CTMC as a tuple $C = (S, s_{init}, \mathbf{R}, AP, L)$ which consists of:

- a countable, non-empty set of states S
- an initial state $s_{init} \in S$
- a transition *rate* matrix $\mathbf{R} : S \times S \longrightarrow \mathbb{R}_{\geq 0}$
- a set of atomic propositions AP, and
- a labeling function $L : S \longrightarrow 2^{AP}$

■

The elements S, s_{init}, AP, and L are as in DTMC. However, the transition rate matrix $\mathbf{R}$ gives the rates as opposed to the probabilities. The probability of taking a transition from state s to state s' within t time units equals $1 - e^{-\mathbf{R}(s,s')\cdot t}$. In the case of the origin state s having more than one successor states s' such that $\mathbf{R}(s, s') > 0$, there exists a *race condition*. Namely, the transition taken will be the one which is enabled first. Therefore, the probability $\mathbf{P}(s, s')$ of moving from state s to state s' in a single transition is $\mathbf{R}(s, s')/E(s)$ where $E(s)$ is called the *exit rate* and calculated as $\sum_{s' \in S} \mathbf{R}(s, s')$. Notice that, in the case of no outgoing transitions from s, $\mathbf{P}(s, s') = 1$ for $s = s'$ and $\mathbf{P}(s, s') = 0$ for $s \neq s'$.

The time spent in a state s before any transition happens (namely the *mean sojourn time*) is exponentially distributed with rate $E(s)$.

The probabilistic information of a CTMC is actually captured by an *embedded DTMC* $emb(C) = (S, s_{init}, \mathbf{P}, AP, L)$ where S, s_{init}, AP, and L are identical to the CTMC and $\mathbf{P}(s, s')$ is as described above. ■

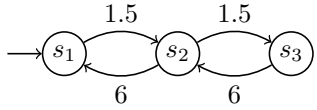
We also need to define another matrix which will be used in the analysis of CTMC. The (infinitesimal) *generator matrix* $\mathbf{Q} : S \times S \longrightarrow \mathbb{R}$ of a CTMC $C = (S, s_{init}, \mathbf{R}, AP, L)$ is defined as:

$$\mathbf{Q}(s, s') = \begin{cases} \mathbf{R}(s, s') & if\, s \neq s' \\ -\sum_{s'' \neq s} \mathbf{R}(s, s'') & otherwise. \end{cases}$$

■

A path through a CTMC is a sequence of states and sojourn times (durations) $\sigma = s_0 t_0 s_1 t_1 \ldots$ where $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t_i \in \mathbb{R}_{\geq 0}$ for all $i \geq 0$. The t_i value represents the amount of time spent in the state s_i. As for DTMCs, $\sigma[i]$ denotes the $(i+1)$th state of σ, i. e., $\sigma[i] = s_i$. In addition, we define $\delta(\sigma, i) = t_i$ as the time spent in state s_1, and we define $\sigma@t = \sigma[i]$ as the state occupied at time t i.e. where i is the smallest index for which $t < \sum_{j=0}^{i} t_j$

Example 4.3 (Continuous-time Markov chain) *Consider a simple model of a queueing system that can have maximum two jobs in the queue. CTMC model of the system can be given in the graphical notation as:*



where in the initial state s_1 no jobs are waiting in the queue, in state s_2 one job arrived and finally in s_3 two jobs are in the queue. The jobs arrive with the rate 1.5 and leave the queue with the rate 6.

The transition rate matrix $\mathbf{R}$ *and (embedded) probability matrix* $\mathbf{P}$ *of this example are:*

$$\mathbf{R} = \begin{pmatrix} 0 & 1.5 & 0 \\ 6 & 0 & 1.5 \\ 0 & 6 & 0 \end{pmatrix} \quad \mathbf{P} = \begin{pmatrix} 0 & 1 & 0 \\ 0.8 & 0 & 0.2 \\ 0 & 1 & 0 \end{pmatrix}$$

Note that, the generator matrix $\mathbf{Q}$ *of this example can be given by modifying* $\mathbf{R}$*'s diagonal to be* $\mathbf{R}_{1,1} = -1.5$*,* $\mathbf{R}_{2,2} = -7.5$*, and* $\mathbf{R}_{3,3} = -6$*.*

As an additional note, this kind of models are known as birth–death processes, a special kind of CTMC where the states represent the current size of a population and the transitions are limited to births and deaths.

■

4.3 Property Specification

In order to make accurate and reliable verification, properties should be described in a precise and unambiguous manner. Although we can describe specific

properties in a suitably rich mathematical logic such as first order logic, we need to limit this expressiveness if we are to *automatically verify* a property. More specifically, the challenge is to find a logic that is both expressive enough for the properties we are interested in, and admits efficient model checking algorithms.

Temporal Logic is a form of modal logic that is suitable to specify properties of information systems. It is actually an extension of traditional propositional logic with operators that refer to the behaviour of systems over time. Temporal logic allows specification of a wide range of system properties such as correctness, reachability, safety, liveness, fairness, etc. Below we briefly explain the main properties that can be specified:

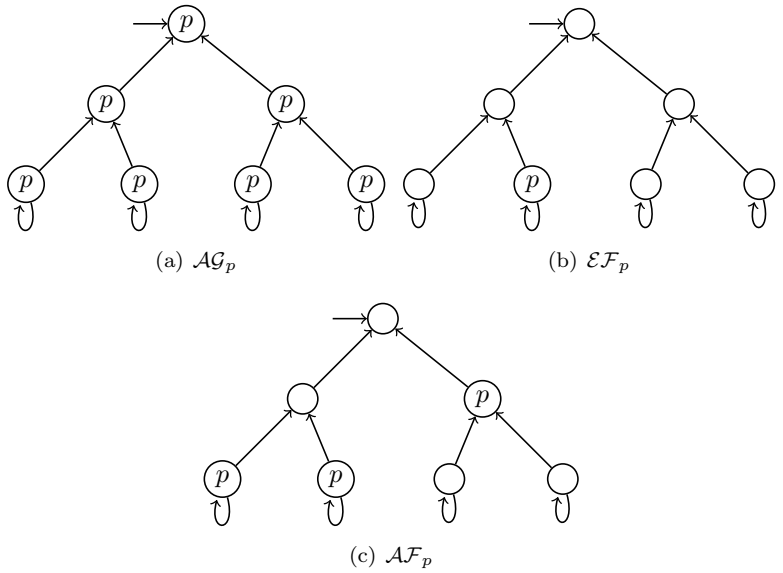
- A *correctness* property states that system does what it is supposed to do.
- A *reachability* property states that it is possible to end up in a desired state.
- A *safety* property states that some bad things never happens.
- A *liveness* property states that some good things eventually happens.
- A (strong) *fairness* property states that something is recurrent.

As we mentioned, temporal logic is a formalism for describing change over time, and was suggested to be used for reasoning about concurrent programs by Pnueli in [Pnu77]. If a program can be specified in temporal logic, then it can be realized as a finite state system. This actually suggested the idea of checking if a finite state graph is a model of a temporal logic specification, which is actually the idea of *model checking*.

Pnueli used a temporal logic with basic temporal operators $\mathcal{F}$ (**F**inally, sometime) and $\mathcal{G}$ (**G**lobally, always). Augmented with $\mathcal{X}$ (ne**X**t) and $\mathcal{U}$ (**U**ntil), this is today known as Linear Temporal Logic (LTL) [MP92].

Another widely used logic is the Computation Tree Logic (CTL) [CJ97]. The basic temporal modalities in CTL are $\mathcal{A}$ (**A**ll, for all futures) or $\mathcal{E}$ (**E**xists, for some future) followed by one of $\mathcal{F}$, $\mathcal{G}$, $\mathcal{X}$, and $\mathcal{U}$. Compound formulae of CTL are built up from nesting and propositional combination of CTL subformulae.

CTL is a branching time logic as it can distinguish between $\mathcal{A}\mathcal{F}_p$ (along all futures, p eventually holds and is thus inevitable) and $\mathcal{E}\mathcal{F}_p$ (along some future, p eventually holds and is thus possible). The difference between CTL and LTL is in the treatment of time, i.e. an LTL formula refers to a specific execution path



(a) $\mathcal{AG}_p$ (b) $\mathcal{EF}_p$

(c) $\mathcal{AF}_p$

Figure 4.1: Example Usage of Temporal Operators

in the model, whereas a CTL formula refers to a tree of possible computations. Examples for the use of temporal operators are given in Fig. 4.1.

Temporal logic formulae are traditionally interpreted over a given *Kripke* structure, that is basically a finite state graph. A Kripke structure $M = (S, I, R, L)$ is a 4-tuple that consists of:

- a finite set of states S
- a set of initial states $I \subset S$
- a binary transition relation $R \subseteq S \times S$
- a labeling function (states to atomic propositions) $L : S \to 2^{AP}$

Then, we may write $M, s_0 \models p$ to state that in Kripke structure M, at state s_0, formula p is true. ■

For a model with n states and m transitions, and a property Φ, the complexity of CTL model checking is $O((n+m)|\Phi|)$, whereas LTL model checking is $O((n+m)2^{|\Phi|})$. On the other hand, it is generally considered to be more intuitive to specify properties in LTL.

The logics LTL and CTL have turned out to be very powerful and influential, both in academic and industrial use. Examples of prominent industrial logics, mostly used in hardware verification, include *IBM Sugar* based on CTL, *Intel ForSpec* based on LTL, etc.

A number of logics have been developed that extends CTL. The fact that branching time was favoured over linear time is most likely because of the lower complexity of the model checking problem.

The semantics and model checking algorithms of both CTL and LTL can naturally be extended to a probabilistic setting. Since there are an infinite number of computation paths in any non-trivial model, we need to assign probabilities to *measurable sets of paths*. In CTL, computation trees naturally form measurable sets of paths (so-called *cylinder sets*), and since either the satisfaction or violation of a given LTL formula can be demonstrated by a finite prefix of a path, we have a similar construction in both cases.

Both explicit state and symbolic model checking algorithms have been extended to probabilistic systems. In the case of explicit state model checking, graph reachability algorithms are essentially modified to solve *probabilistic* reachability problems. In the case of symbolic model checking, Binary Decision Diagrams (BDD) are extended to Multi-Terminal Binary Decision Diagrams (MTBDD), which are efficient data structures for representing *real-valued* functions (i.e. those that map to a probability), rather than just Boolean functions.

In the following sections, we will describe merely the logics which are both relevant to our study and also widely used in practice. We start with continuous stochastic logic in Section 4.3.1. Then, we describe CSRL as the variant of CSL augmented with rewards in Section 4.3.2.

4.3.1 CSL

Continuous Stochastic Logic (**CSL**) [ASSB96, BHHK00] is a logic for expressing properties of CTMCs, and has the following syntax:

$$\begin{array}{lcl} \Phi & ::= & \texttt{true} \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\trianglelefteq p}(\varphi) \mid \mathcal{S}_{\trianglelefteq p}(\Phi) \\ \varphi & ::= & \mathcal{X}\,\Phi \mid \Phi\,\mathcal{U}\,\Phi \mid \Phi\,\mathcal{U}^{I}\,\Phi \end{array}$$

We start with the distinction between *state formulae* and *path formulae*. A state formula holds of states in the model, whereas a path formula holds of (possibly infinite) sequences of states, namely paths. We use Φ to denote state formulae, and φ to denote path formulae.

Then comes a propositional fragment where $a \in AP$ is an atomic proposition, or label of a state.

The probability measure formula $\mathcal{P}_{\trianglelefteq p}(\varphi)$ states that the probability measure over paths that satisfy φ is $\trianglelefteq p$, where $\trianglelefteq \in \{<, \leq, \geq, >\}$.

The formula $\mathcal{S}_{\trianglelefteq p}(\Phi)$ asserts that the steady-state probability of being in a state satisfying Φ meets the bound $\trianglelefteq p$.

$\mathcal{X}\,\Phi$ is the untimed next operator, which holds for a path σ if the next state satisfies Φ. $\Phi_1\,\mathcal{U}\,\Phi_2$ is the untimed until operator, which holds for a path σ if some state in the future satisfies Φ_2, and all states before this point satisfy Φ_1.

Note that various commonly-used operators (and also used in this thesis) can be derived using until formula as follows:

$$\begin{aligned} \mathcal{F}\,\Phi &= \mathtt{true}\,\mathcal{U}\,\Phi \\ \mathcal{G}\,\Phi &= \neg(\mathtt{true}\,\mathcal{U}\,\neg\Phi) \end{aligned}$$

Before proceeding to semantics, we need to formally define a *path*. A path σ in a CTMC is a (possibly infinite) alternating sequence of states and durations $s_0, t_0, s_1, t_1, \ldots$, such that $s_i \in S$, $t_i \in \mathbb{R}_{>0}$, and for all $i < |\sigma|-1$, $\boldsymbol{P}(s_i, s_{i+1}) > 0$ and $r(s_i) > 0$. We write $\sigma[i] = s_i$ and define $Paths(s)$ to be the set of all (infinite and finite) paths of a CTMC starting in state s, i.e $\sigma[0] = s$. We additionally define $\delta(\sigma, i) = t_i$ as the time spent in state s_i, and $\sigma@t = \sigma[i]$ as the state occupied at time t. ■

The semantics of CSL is as follows. $s \models \Phi$ means that a state s satisfies a state formula Φ, and $\sigma \models \varphi$ means that a path σ satisfies a path formula φ.

s	$\models$	$\mathtt{true}$	for all	$s \in S$
s	$\models$	a	iff	$a \in L(s)$
s	$\models$	$\Phi_1 \wedge \Phi_2$	iff	$s \models \Phi_1$ and $s \models \Phi_2$
s	$\models$	$\neg\Phi$	iff	$s \not\models \Phi$
s	$\models$	$\mathcal{P}_{\trianglelefteq p}(\varphi)$	iff	$\Pr\{\, \sigma \in Paths(s) \mid \sigma \models \varphi \,\} \trianglelefteq p$
s	$\models$	$\mathcal{S}_{\trianglelefteq p}(\Phi)$	iff	$\lim_{t\to\infty} \Pr\{\, \sigma \in Paths(s) \mid \sigma@t \models \Phi \,\} \trianglelefteq p$
σ	$\models$	$\mathcal{X}\,\Phi$	iff	$\sigma[1] \models \Phi$
σ	$\models$	$\Phi_1\,\mathcal{U}^I\,\Phi_2$	iff	$\exists t \in I.\ \sigma@t \models \Phi_2$ and $\forall t' < t.\ \sigma@t' \models \Phi_1$

Since CSL is a continuous time logic, the intervals on the path operators can be real-valued.

As an example of the sort of properties we can express, consider the following CSL formula, for $AP = \{ \mathit{breakdown}, \mathit{recovery} \}$:

$$\mathcal{P}_{\geq 0.75}(\neg \mathit{breakdown}\, \mathcal{U}^{[0,10]}\, \mathit{recovery})$$

which specifies that when a breakdown occurs, the probability of system being recovered in the first 10 minutes is at least 0.75. Note that the time unit is implicit to the model.

4.3.2 CSRL

We can add reward structures to CTMCs such that the reward assigned to a state is not a fixed reward that is given for each time step we occupy that state, but a *rate* of reward acquisition.

To reason about reward-structured CTMCs, the Continuous Stochastic Reward Logic (**CSRL**) [CKKP05] was developed as an extension of CSL. It has the following syntax:

$$\begin{array}{lcl} \Phi & ::= & \mathtt{tt} \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\trianglelefteq p}(\varphi) \mid \mathcal{S}_{\trianglelefteq p}(\Phi) \\ \varphi & ::= & \mathcal{X}_J^I\, \Phi \mid \Phi\, \mathcal{U}_J^I\, \Phi \end{array}$$

Before explaining the extension to CSL, we would like to extend the next operator since we would use the extended version in CSRL. We additionally define a *time-bounded next operator* as presented in [BKHW05]. The path formula $\mathcal{X}^I\, \Phi$ requires that the next state satisfies Φ, *and* that the transition will take place in the time interval I. The semantics of the timed next operator is as follows:

$$\sigma \models \mathcal{X}^I\, \Phi \text{ iff } \sigma[1] \models \Phi \text{ and } \delta(\sigma, 0) \in I$$

The only extension to CSL is the addition of the time-bounded and reward-bounded next and until operators. $\mathcal{X}_J^I\, \Phi$ states that the next state satisfies Φ, and the transition is made at some time $t \in I$, and the accumulated reward until time t is in the interval J. The until operator $\Phi_1\, \mathcal{U}_J^I\, \Phi_2$ means that Φ_2 holds within a time interval $n \in I$, that all states before this satisfy Φ_1, and that the accumulated reward before satisfying Φ_2 is in the interval J. Note that we use a superscript to talk about time, and a subscript to talk about rewards.

The semantics of CSRL is interpreted over a Continuous time Markov Reward Model (CMRM). This is a tuple $(S, s_{init}, \mathbf{R}, AP, L, \rho, \iota)$, where $(S, s_{init}, \mathbf{R}, AP, L)$ is a CTMC, $\rho : S \to \mathbb{R}_{\geq 0}$ is a reward structure describing the *rate* per time unit at which a reward is accumulated in each state, and $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a reward

structure describing the *impulse reward* accumulated when a transition is made between two states. It must be the case that for all $s \in S$, $\iota(s,s) = 0$. We define the accumulated reward along a path σ at time t to be:

$$y_\sigma(t) = \rho(\sigma@t).\left(t - \sum_{j=0}^{i-1} \delta(\sigma, j)\right) + \sum_{j=0}^{i-1} \rho(\sigma[j]).\delta(\sigma, j) + \sum_{j=0}^{i-1} \iota(\sigma[j], \sigma[j+1])$$

■

The semantics of the new operators in CSRL is then as follows:

$$\begin{array}{lcll} \sigma & \models & \mathcal{X}_J^I \Phi & \text{iff} \quad \sigma[1] \models \Phi \text{ and } \delta(\sigma,0) \in I \text{ and } y_\sigma(\delta(\sigma,0)) \in J \\ \sigma & \models & \Phi_1 \mathcal{U}_J^I \Phi_2 & \text{iff} \quad \exists t \in I.\ \sigma@t \models \Phi_2 \text{ and } \forall t' < t.\ \sigma@t' \models \Phi_1 \\ & & & \qquad \text{and } y_\sigma(t) \in J \end{array}$$

4.4 Model Checking

In this section, we first present algorithms for CSL model checking and then present a well-established model checker that we use in the following chapters and implements CSL model checking algorithms. Thus we will complete the last phase of model checking, after modelling and property specification.

4.4.1 CSL Model Checking over CTMCs

In this section, we consider a CSL model checking algorithms over CTMCs from [KNP07].

The algorithms take a CTMC $C = (S, s_{init}, \mathbf{R}, AP, L)$ and a CSL formula Φ, and output a set of states $Sat(\Phi) = \{s \in S \mid s \models \Phi\}$ that satisfies Φ.

First, a parse tree of Φ is constructed. Then, the set of states that satisfying each subformula is recursively computed upwards towards the root of the tree. In the end, for each state it is determined whether it satisfies Φ or not.

4.4.1.1 Probability Measure

In Section 4.3.1 we have defined *path measures*, now we should define the *probability measure* that we will use in the following sections. Sticking to the def-

initions from [KNP07] and omitting all the details, we define the probability measure Pr_s as a unique measure such that $Pr_s(\mathcal{C}(s)) = 1$ and $Pr_s(\mathcal{C}(s, I, \ldots, I_{n-1}, s_n, I', s'))$ is computed as:

$$Pr_s(\mathcal{C}(s, I, \ldots, I_{n-1}, s_n)) \cdot \mathbf{P}(s_n, s') \cdot \left(e^{-E(s_n).inf I'} - e^{-E(s_n).inf I'}\right)$$

where $\mathcal{C}(\sigma)$ is a *cylinder set* i.e. the set of all infinite paths with the finite path σ as prefix. Notice that, $\mathbf{P}$ is the transition probability matrix of the *embedded DTMC* and $E(s)$ is the *exit rate* of state s as we defined in Section 4.3.1.

4.4.1.2 Transient and Steady–state Probabilities

At this point, we would like to discuss two traditional properties of CTMCs: *transient behaviour* and *steady-state behaviour*.

The transient behaviour relates to the state of the model *at a particular time instant*. The transient probability on a CTMC

$$\pi_{s,t}(s') = Pr_s\{\sigma \in Paths(s) \mid \sigma@t = s'\}$$

is defined as the probability of being in state s' at time instant t. Notice that s is the starting state, $\sigma@t$ is the state occupied at time t for the path σ, and $Paths(s)$ is the set of all paths of the considered CTMC starting from state s, as we defined in Section 4.3.1. ■

The steady-state behaviour relates to the state of the model *in the long run*. The steady-state probability on a CTMC

$$\pi_s(s') = \lim_{t \longrightarrow \infty} \pi_{s,t}(s')$$

is defined as the probability of having started in state s, being in a state s' in the long run.

The values $\pi_s(s')$ for all $s \in S$ describe the steady-state probability distribution, which can be used for inferring the percentage of time that is spent in each state in the long run. ■

4.4.1.3 Uniformisation

In this section, we introduce *uniformisation* method which is used for computing transient probabilities, and relied on in the CSL model checking algorithms over CTMCs.

The key idea in this technique is, to compute the transient probabilities we work on the *uniformised DTMC* of the CTMC, where each step corresponds to one exponentially distributed delay. Now we introduce the transition probability matrix $\mathbf{P}^{unif}$ for the uniformised DTMC as

$$\mathbf{P}^{unif} = \mathbf{I} + \mathbf{Q}/q$$

such that the *uniformisation rate* q should satisfy

$$q \geq max\{E(s)|s \in S\}$$

■

At this point, we introduce another matrix that we will use in our computations. The matrix of all transient probabilities for time t is defined as $\mathbf{\Pi_t}$, such that $\mathbf{\Pi_t}(s, s')$ is the probability, having started in state s, of being in state s' at time instant t. In [Ste94], it is shown that $\mathbf{\Pi_t}$ can be expressed as a matrix exponential such that $\mathbf{\Pi_t} = e^{\mathbf{Q}.t}$ and therefore can be evaluated as a power series:

$$\mathbf{\Pi_t} = \sum_{i=0}^{\infty} \frac{(\mathbf{Q}.t)^i}{i!}$$

where $\mathbf{Q}$ is the *generator matrix* that we defined in Section 4.2.2.2.

However, this computation tends to be unstable, therefore the probabilities are computed through the uniformised DTMC $\mathbf{P}^{unif}$ instead of CTMC. This brings us to the equality:

$$\mathbf{\Pi_t} = \sum_{i=0}^{\infty} \gamma_{i,qt} \quad (\mathbf{P}^{unif})^i \quad \text{where}\, \gamma_{i,qt} = e^{-qt} \frac{(qt)^i}{i!}$$

where $(\mathbf{P}^{unif})^i$ is the probability of jumping between each pair of states in i steps, and $\gamma_{i,qt}$ is the (Poisson) probability of i such steps occurring in time t, given that the delay is exponentially distributed with rate q. The computation of the infinite sum above is out of our scope for preliminaries. Note that, (unlike $\mathbf{Q}$) the matrix $\mathbf{P}^{unif}$ is **stochastic**, *i.e.* all entries are in the range [0,1] and all rows sum to 1.

4.4.1.4 Algorithms

Algorithms for stochastic model checking derive from conventional model checking techniques, together with mathematical techniques from linear algebra and Markov chains. In this thesis, we consider model checking algorithms for CSL property specification over CTMC models. Model checking problem for CSL was shown to be decidable (for rational time bounds) in [ASSB96]. Approximate model checking algorithms have been studied in [BKH99] where the satisfaction of time-bounded until formulas is shown to be based on solving a type of integral equation system. The probabilistic model checker that we use in this thesis, PRISM, implements algorithms based on this work and further improvements in [KKNP01, BHHK03]. The details on the CSL model checking algorithms used in PRISM are present in [KNP07] and they are out of our scope for preliminaries.

However, we should mention that the overall time complexity for model checking a CSL formula against a CTMC C is linear in the logic formula, polynomial in the set of states and linear in the product of the uniformisation rate (defined as q in the previous section) and the maximum t value found in the parameter of a time-bounded until operator. Note that the size of a logic formula is equal to the number of logical connectives and temporal operators in the formula plus the sum of the sizes of the temporal operators.

4.4.2 PRISM

The probabilistic symbolic model checker PRISM [KNP07, Pria] is being maintained and developed by Marta Kwiatkowska's research group at Oxford University. PRISM provides direct support for various Markov models, yet we will only cover the necessary fragment of PRISM that we need in our developments in the next chapters (i.e. CTMC models and CSL model checking).

PRISM is a *symbolic* model checker since in the sense that it uses data structures based on binary decision diagrams (BDDs). BDDs provide compact representations by exploiting regularity. More specifically, PRISM uses multi–terminal BDDs (MTBDDs) which is an efficient way of representing very large models with numerical values.

The numerical computation of model checking algorithms are based around three numerical engines:

- The *sparse engine* is using sparse matrix representations of the model, and

corresponds to explicit state model checking. Typically, sparse engine is faster than MTBDD engine but requires more memory.

- The *MTBDD engine* is using MTBDDs to represent a model, and corresponds to probabilistic symbolic model checking.
- The default engine, *hybrid engine*, uses a combination of the above two representations [KNP04]. The transition matrix of the model is stored in an MTBDD, whereas the iteration vector that recording the probability of each state satisfying a given property is stored as a full array. This gives better results than the MTBDD engine for most models, since the majority of states have different probabilities of a satisfying a given property, meaning that the iteration vector cannot be stored efficiently as an MTBDD.

We will first give an overview of modelling using PRISM language, that maps onto a CTMC in our case and enriched with reward structures. We will then describe the PRISM property specification language.

4.4.2.1 Modelling in PRISM

The PRISM modelling language [Prib] is a state-based language based on the Reactive Modules formalism [AH96]. It has two main components: *variables* and *modules*. A PRISM model consists of a number of modules that run in parallel. Each contains local variables that constitute its state and a set of *guarded commands* that describe its behaviour. Global variables are also allowed, which can be read and modified by all modules.

The behaviour of a PRISM module is described by a set of guarded commands. A guarded command has the following general form:

$$[\,]\ g \rightarrow \lambda_1 : u_1 + \cdots + \lambda_n : u_n$$

The guard g is a condition on the state of the variables, i.e. it determines a set of states in which the command can execute. If g is true, then with probabilistic information λ_i (which is a *rate* in CTMC), update u_i is performed. An update specifies how the state of the local variables changes (we write x to refer to the old state of a variable and x' to refer to the new state).

As a primitive example, consider the simple CTMC that we sketched in Example 4.3. This CTMC that we visualized in state-transition diagram can be modelled in PRISM as listed in Table 4.1. There, we have a single module

```
ctmc

module example_queue_system

   s: [1..3] init 1;

   [ arrive ] s < 3   ->      1.5 : (s'=s+1);
   [ serve ]  s > 1   ->        6 : (s'=s-1);

endmodule
```

Table 4.1: PRISM model of *Example 4.3*

`example_queue_system` with a single local variable `s`, and two guarded commands.

PRISM builds models by taking composition of the modules, such that commands synchronise over *action*s. Actions are placed between the square brackets at the beginning of the commands. In Table 4.1 we defined two actions `arrive` and `serve` and in Table 4.2 we defined a client *module* that can synchronise over those actions. Notice that, we did not specify an arrival rate nor a leave rate for the client, so that those rates will be equal to 1 (default) and multiplied with the rates on the corresponding actions in `example_queue_system`. Besides, we skipped the model type `ctmc` at the beginning since it is not a model but a module only.

```
module client_1

   c1_queued: bool init false;

   [ arrive ]  c1_queued = false -> (c1_queued'=true);
   [ serve ]   c1_queued = true  -> (c1_queued'=false);

endmodule
```

Table 4.2: PRISM model of a client

A naive parallel composition of the modules `example_queue_system` and `client_1` would yield six states since we have three possible values for the variable `s`, and two possible values for the Boolean variable `c1_queued`. However, only two of those states are *reachable*: $\{\texttt{s} = 1, \texttt{c1\_queued} = \texttt{false}\}$ (the initial state), and $\{\texttt{s} = 2, \texttt{c1\_queued} = \texttt{true}\}$.

However, if we add another client `client_2` to the system, such that it is identical to `client_1` but has the variable `c2_queued` instead of `c1_queued`, and we synchronise each clients with the queue system things will change. Below is the line of code that we need to add to our module in order to provide synchronous parallel composition of clients and the queue, and asynchronous parallel composition of clients:

```
system (client_1 ||| client_2) || example_queue_system endsystem
```

Now there will be four reachable states out of twelve:
$\{\mathtt{s} = 1, \mathtt{c1\_queued} = \mathtt{false}, \mathtt{c2\_queued} = \mathtt{false}\}$ (the initial state),
$\{\mathtt{s} = 2, \mathtt{c1\_queued} = \mathtt{false}, \mathtt{c2\_queued} = \mathtt{true}\}$,
$\{\mathtt{s} = 2, \mathtt{c1\_queued} = \mathtt{true}, \mathtt{c2\_queued} = \mathtt{false}\}$,
and $\{\mathtt{s} = 3, \mathtt{c1\_queued} = \mathtt{true}, \mathtt{c2\_queued} = \mathtt{true}\}$.

For a CTMC model, the commands take place at a particular *rate*, rather than with a certain probability. When PRISM takes the composition of the modules, every enabled command in a state of the system is allowed to proceed at the specified rate. Intuitively, this means that the exit rate from a state of the system is the *sum* of the exit rates of the individual states, and the probability of one command proceeding over another depends on the relative exit rate.

Another important PRISM feature in modelling is the *reward structures*. Each model can be extended with reward structures, such that each reward structure is defined separately outside any module definitions. A reward item has the following general forms:

$$g \ : r; \quad \text{for state rewards}$$

$$[\ a\ ]\ g : r; \quad \text{for transition rewards}$$

where g is a predicate, a is an action, and r is a real-valued expression.

As a simple example, the reward structure `reward_1` below assigns a transition reward of 1 to transitions labelled `serve` from states satisfying `c1_queued = true`.

```
rewards "reward_1"
   [ serve ] c1_queued = true :  1
endrewards
```

4.4.2.2 Property Specification

In this section, we will consider the properties in PRISM that are based on two logics we described in Section 4.3.2 and Section 4.3.1.

There are three operators that PRISM provides for state formulae — `P` for the probability of satisfying a path formula, `S` for the long-run or steady-state probability of being in a certain set of states, and `R` for reward properties. The `P` and `S` operators have the following syntax, where Φ is a state formula and φ is a path formula:

`P` $\trianglelefteq p$ `[` φ `]`	`S` $\trianglelefteq p$ `[` Φ `]`	`R` $\trianglelefteq r$ `[` ρ `]`
`P =?` `[` φ `]`	`S =?` `[` Φ `]`	`R =?` `[` ρ `]`

As you can see above, not only the *bound*s $\trianglelefteq p$ and $\trianglelefteq r$ can be used with the operators, but also a *query* can be specified using =?. This way we can directly specify properties which evaluate to a numerical value. since the model checking algorithms proceed by computing the actual probabilities and then composing it to the bound, no additional computation is needed for the operators with =?. Besides, numerical values are very useful from a practical standpoint.

The reward property, ρ, has four different types:

- **Reachability reward:** `F` Φ
  Reachability reward property specifies the expected reward accumulated along a path until a state satisfying Φ is reached.

- **Cumulative reward:** `C <=` t
  Cumulative reward property specifies the expected reward accumulated along a path until a time bound t, which is a real number for CTMC models.

- **Instantaneous reward:** `I =` t
  Instantaneous reward property specifies the expected reward of the model at a time instant t.

- **Steady-state reward:** `S`.
  Steady-state reward property specifies the reward per time unit in the long run.

4.5 Bisimulation

Bisimulation is a binary relation between state-transition systems, that associates systems which behave in the same way. In other words, we can find out if a system simulates the other and vice-versa. Intuitively two systems are *bisimulation-equivalent* or *bisimilar* if they match each other's moves. In this sense, the systems cannot be distinguished from each other by an observer.

In this section, we will mostly follow the definitions and notation from [KKZJ07] and [BK08] and build the definitions in a coherent way with the previous sections.

4.5.1 Bisimulation Equivalence – as a relation between transition systems

We will start by defining bisimulation for labelled transition systems. Let $LTS_1 = (S_1, I_1, \longrightarrow_1, Act_1, AP, L_1)$ and $LTS_2 = (S_2, I_2, \longrightarrow_2, Act_2, AP, L_2)$ be two labelled transition systems over a set of atomic propositions AP. A *bisimulation* for (LTS_1, LTS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that:

- $\forall s_1 \in I_1(\exists s_2 \in I_2.(s_1, s_2) \in \mathcal{R})$ and $\forall s_2 \in I_2(\exists s_1 \in I_1.(s_1, s_2) \in \mathcal{R})$
- for all $(s_1, s_2) \in \mathcal{R}$ it holds:
  - $L_1(s_1) = L_2(s_2)$
  - if $s_1' \in \sigma(s_1)$ then there exists $s_2' \in \sigma(s_2)$ with $(s_1', s_2') \in \mathcal{R}$
  - if $s_2' \in \sigma(s_2)$ then there exists $s_1' \in \sigma(s_1)$ with $(s_1', s_2') \in \mathcal{R}$

LTS_1 and LTS_2 are *bisimulation-equivalent* or shortly *bisimilar*, denoted by $LTS_1 \sim LTS_2$, if there exists a bisimulation $\mathcal{R}$ for LTS_1 and LTS_2. ■

Thus, we required that every initial state of the two LTSs be related to each other, states s_1 and s_2 in a pair $(s_1, s_2) \in \mathcal{R}$ are equally labelled, and every outgoing transition of s_1 must be matched by an outgoing transition of s_2 and vice versa.

4.5.2 Bisimulation Equivalence – as a relation on states

An alternative perspective on bisimulation is to consider bisimulation relation between states in a single transition system, rather than a relation between transition systems. This sense of bisimulation can be used in obtaining smaller models out of large models. Let $LTS = (S, I, \longrightarrow, Act, AP, L)$ be a labelled transition system, a *bisimulation* for LTS is a binary relation $\mathcal{R}$ on S such that for all $(s_1, s_2) \in \mathcal{R}$:

- $L(s_1) = L(s_2)$
- if $s_1' \in \sigma(s_1)$ then there exists an $s_2' \in \sigma(s_2)$ with $(s_1', s_2') \in \mathcal{R}$
- if $s_2' \in \sigma(s_2)$ then there exists an $s_1' \in \sigma(s_1)$ with $(s_1', s_2') \in \mathcal{R}$

The states s_1 and s_2 are *bisimulation-equivalent* or shortly *bisimilar*, denoted by $s_1 \sim s_2$, if there exists a bisimulation $\mathcal{R}$ for LTS with $(s_1, s_2) \in \mathcal{R}$. ■

These two definitions of bisimulation reveals that, a bisimulation on the states for an LTS is a bisimulation on transition systems for the pair (LTS,LTS) except that the first condition (for bisimulation as a relation between transition systems), i.e. $\forall s_1 \in I_1(\exists s_2 \in I_2.(s_1, s_2) \in \mathcal{R})$ and $\forall s_2 \in I_2(\exists s_1 \in I_1.(s_1, s_2) \in \mathcal{R})$, is not required.

4.5.3 Bisimulation In Probabilistic Context

In the purely probabilistic context, the study of strong bisimulation was initiated by Larsen and Skou [LS91], and an equivalence notion was developed, similar to the queuing theory notion of *lumpability* [KS60]. We will start with a definition of strong bisimulation for DTMCs taken from [KKZJ07]. Let $D = (S, \mathbf{P}, L)$ be a DTMC, and R be an equivalence relation on S.

Let $D = (S, \mathbf{P}, L)$ be a DTMC and R an equivalence relation on S. The quotient of S under R is denoted S/R. R is a *strong bisimulation* on D if for $s_1 R s_2$:

$$L(s_1) = L(s_2) \text{ and } \mathbf{P}(s_1, C) = \mathbf{P}(s_2, C) \text{ for all } C \text{ in } S/R.$$

s_1 and s_2 in D are *strongly bisimilar*, denoted $s_1 \sim_d s_2$, if there exists a strong bisimulation R on D with $s_1 R s_2$.

Strong bisimulation for CTMCs [BRNN04], that implies ordinary lumpability, is a mild variant of the notion for the discrete-time probabilistic setting: in addition to the above, it is also required that the exit rates of bisimilar states are equal: $E(s_1) = E(s_2)$.

The standard model checking algorithm for the CSL time-bounded until operator is based upon uniformisation — we have to consider the three cases of $[0, t]$, $[t, \infty]$, and $[t_1, t_2]$ for the time interval, but the algorithm is essentially a first passage time analysis [BHHK03]. For the timed next operator, the model checking algorithm boils down to a matrix-vector multiplication.

One problem we face with CTMCs is that uniformisation does *not* preserve the validity of all CSL formulae. This is a problem if we perform lumpability based abstractions, since these are typically based on a uniformised CTMC [KKLW07]. We write CSL$\backslash X$ to mean the subset of CSL without the next operator. If two CTMCs are weakly bisimilar, then the validity of all CSL$\backslash X$ formulae is preserved [BKHW05]. A consequence is that the uniformisation of a CTMC preserves CSL$\backslash X$ equivalence.

Chapter 5

Modelling Scenarios

Protection of data in transit, such as in networks, is often provided by encryption algorithms that rely on cryptographic keys. Algorithms are assumed to be known by the attackers, whereas *only secrecy of the key provides security* as stated in Kerckhoffs' principle [Ker83]. However, keys may get compromised over the time, therefore networks secured by encryption usually have the notion of *key update* where the current key is revoked and a new key is established.

Today it is a well-known precaution that the cryptographic keys should be updated over time, while it is still not clear *how* and *when* the keys should be updated. In this chapter, we model key update strategies on the running example of ZigBee wireless sensor networks.

In Section 5.1, we define the problem that we want to solve and give clues on our approach. In Section 5.2, we explain the basics of the scenarios to be modelled, which include network security details, key update strategies, and application profiles in our running example. In Section 5.3, we present our work towards developing a formal model that is both realistic and scalable. We gradually refine and improve the model to be suitable for analyses in the following chapters.

5.1 Problem and Solution Approach

We start by describing a generic scenario on ZigBee wireless sensor networks. We consider a network where devices can *leave* the network (e.g. having a breakdown, drained batteries, relocation etc.), and new devices can *join* the network (e.g. enlarging the network to benefit from more services, device replacement, etc.). The communication within the network is secured by symmetric encryption, such that each device in the network shares the same key because of low-resource nature of the sensor networks.

When a new device joins a ZigBee network it will register with the trust center and store the current valid cryptographic key in its memory (regardless of possessing the key before joining). But what happens when a device leaves the network? There is a risk that its memory still contains the key. In the worst case this means that in the hands of a dishonest principal the key may be used to manipulate the records of our energy consumption, it may be used to break the security of our homes or it may be used to interfere with the medical equipment of our patients. Surely we want to avoid that!

There seems only one way to achieve this, so the answer seems straightforward:

Change the cryptographic key whenever a device leaves the network!

The problem, however, is that the devices are designed to have low power consumption and therefore have very small batteries. Changing the cryptographic key is relatively demanding on power and if done too often it is likely to drain the batteries of the devices. At the same time replacing the batteries is a costly operation - if at all possible - so one would like to avoid that. Thus from this point of view we would argue:

Avoid changing the cryptographic key of the devices to save power!

How can we balance these two viewpoints? When should we update the cryptographic key? Should we update it at regular time intervals, as e.g. once a year? Or should we update it whenever a fixed number of devices, e.g. 10, have left the network? And what difference does it make?

It is essential to strike an acceptable balance: changing the cryptographic key too seldom might jeopardize our homes whereas changing it too often will be a waste of resources in a setting where energy consciousness is a must.

Our approach for solving this problem is as follows: We first build a *model* of the system – in this case a network with a number of sensor devices that with a rate may join and leave the network, as well as send messages over the network. In the cases of leave and message event, there is a risk that the cryptographic key might be compromised. The model also implement a number of strategies for updating the key so that we can investigate the consequences of different choices.

The model should be nicely reflecting and formalizing the properties of the system – i.e. key update in ZigBee networks – and also be scalable to allow quantitative analysis. Having the model the next step is to *analyse* it by posing a number of questions shedding light on its behaviour. We have no way of knowing exactly when the cryptographic key is compromised so we can only investigate the probability for it being compromised.

Throughout this chapter we describe how we develop efficient models for analysis, and we leave the analysis itself to the next chapter.

5.2 Setting the Scene

As we have explained in previous chapters, ZigBee-2007 specifies a suite of security services that includes methods for key establishment, key transport, etc. Although each revision and supporting specifications (such as stack and application profiles) roll out improvements, *key update* strategies and proper determination of related security parameters still remain as gaps in the standard [YNN08].

Naturally, we want to ensure that the risk of using a compromised security key in a ZigBee network is as small as possible – and this calls for updating the key fairly often. On the other hand, this operation is computationally expensive and we would not like to perform it too often. Unfortunately, the ZigBee specification does not give any advice on this (i.e. how and when the key shall be updated) – it merely states that the security key shall be updated (see Appendix B for details).

Given the usual resource limitations in the ZigBee networks, absolute security is often less important than quantifiable trade-offs between security and performance. As security is a qualitative concept, realistic analyses require results that are valid with respect to the full behaviour of the systems considered.

We start by reminding the key points that are necessary for a clear under-

standing of the development, and we omit all the details which are irrelevant to this part of study. ZigBee uses symmetric encryption, the *Advanced Encryption Standard* (AES – 128) [Fed01], therefore all the cryptographic security keys are symmetric keys and 128 bits in length. A *Network Key* (NK) is the mere mandatory key in a ZigBee network, which is shared amongst all the devices and used to secure broadcast communications. A *Trust Center* (TC), creates and distributes the NKs. TC is an application running on a ZigBee device, that is unique in every ZigBee network. As a key component of ZigBee security, the TC is assumed to run on a more powerful device (*e.g.* a coordinator) rather than a regular ZigBee end device. Two more types of security keys may exist in a ZigBee network depending on the security configuration: *Master Key* and *Link Key*. Unlike NK, those keys are pairwise shared.

Network keying scheme (i.e. using NK) has advantages over pairwise keying schemes (i.e. using MK and LK). It uses minimum resources, besides it is simple and easy to manage. There exists benefits in also self-organization and accessibility since neighbouring devices can interpret each other's data. Finally, scalability and flexibility are very high since keying material is the same for all devices.

On the other hand, pairwise keying is not scalable and requires too much storage, energy and computation abilities for low-resource devices. However, using NK is challenging since compromise of one device compromises the entire network. In this study we focus on NK as the key type and refer to it as the *key*, and we assume if a device is in the network then it has already acquired the *key*.

The details of the *NK update* protocol (which is not about any key update strategy but about how to transport the updated key to the devices) is given in the specification and explained in [YNN08]. TC is fully responsible of creating and distributing the NK, and we assume that when TC updates the key, all the devices in the network successfully update their keys. Compromise of a NK affects all the devices in a network.

5.2.1 Key Update Strategies

In the network security area, key update is often seen as a *periodical* event. As such, the ZigBee-2007 specification uses the word "*periodically*" when referring to the key update issue but gives no further guidelines or clues about alternatives. We propose new key update methods for ZigBee wireless sensor networks. Although our starting point is wireless sensor networks, the key update problem exists in almost all kinds of networks that are using encryption, and our methods can be applied in different types of networks as well. In this section we

present the first two methods we propose, together with the classical periodical key update method:

Time-based key update. The *time-based* key update strategy is built on the traditional concept of updating the key after a certain amount of time elapsed. The key is updated after a predefined *key expire time*.

Leave-based key update. The *leave-based* key update strategy is considering the *leave* events in the network. The key is updated after a predefined number of devices leave the network. In practice, when a device leaves the network it may still own a valid key, hence a device leave presents a security risk. To the best of our knowledge, this is a novel key update strategy that we propose in [YNN+10b]. A counter in the TC keeps the number of the devices left (or removed from) the network. When this number reaches the predefined threshold value, all the keys in the network are updated and the counter is reset to zero. The idea here is to have a key update strategy that is inspired by the nature of the wireless sensor networks where number of exchanged messages can be very low but the number of join and leave events can be relatively higher compared to the conventional networks.

Join-based key update. The *join-based* key update strategy is considering the join events in the network. The key is updated after a predefined number of new devices joins the network. A new device that joins the network presents a security risk since it may become a legitimate attacker. This strategy is also our own proposal, and the idea is very similar to the Leave-based key update. The key point here is, even though a device leaving the network with valid key is a risk itself, the bigger risk is when a new device joins after such a leave. Because the new device could have somehow captured the key from a previously left device. In this case, the counter would keep the number of joining devices, and the threshold would be set as a limit for this value.

5.2.2 Application Profiles

As we aim to estimate system dimensions and other parameter values (or value ranges) for different applications, we use application profiles that are defined by ZigBee wireless sensor networks. ZigBee has six different application profiles and up to now only two of them are finalized: *Home Automation* [Zig08a] and *Smart Energy* [Zig08c]. The remaining application profiles that are expected to be finalized and released soon are *Commercial Building Automation*, *Personal, Home and Hospital Care*, *Telecom Applications*, and *Wireless Sensor Applications*. Rather than presenting the details of these profiles, we will explain the properties that are relevant to our work on key updates. This will lead us to

assumptions that we will use in stochastic model checking.

Below we explain the settings for one of the application profiles that we focus on in this study. We leave the explanations for the rest of the application profiles to Appendix B. We first summarize the scope and the purpose of the profile for a better understanding of the design criteria. Then in the second paragraph, we present the information gathered from a professional ZigBee expert that once led the design of ZigBee security sublayer [Cra08], and finally present the parameters that we used in our models with the values that we determined from that information. Specifically, we present the technical details in terms of maximum network size, rate of join, rate of leave, and key compromise probability.

The Home Automation (HA) Profile: This profile covers applications for the residential automation market to allow equipment manufacturers to produce products that will meet the needs of customers ranging from do-it-yourself homeowners to professional installers. Home automation profile has a vast amount of device types such as on/off switch, level control switch, remote control, shade controller, heating cooling unit, various sensors (temperature, pressure, light), intruder alarm system equipments etc.

In this profile, the network is *fairly static* such that it is likely that devices such as light switches and luminaries, once commissioned, would remain in place for a longer period. The network size is in general less than 50 devices. The environment is relatively insecure, and to reflect this we shall say the key is compromised in 1% of the cases. A device may leave the network for reasons such as a break down or flat battery, and most likely, it will be replaced shortly after. We shall assume that each device leaves the network once a year but it will be replaced within a week. Based on these assumptions we specify the remaining constants as follows:

```
maximal size of the network: 20 devices (excluding the trust center)
average number of joining devices: 1 device every week
average number of leaving devices: 1 device per year
risk of key compromise: 1/100
```

Obviously, application profiles can be customized easily and so the models can also be used for different type of networks.

5.3 Developing A Stochastic Model

Developing a formal model for any system is not a trivial step in model checking since a tiny flaw in the model would cause incorrect results in the end. In addition, a slight redundancy in the model can cause a huge increment in the state space which would make model checking infeasible in terms of either time or memory or both.

Our aim for the construction of the model is to make it as simple, modular, and generic as possible, yet detailed enough to allow realistic analysis and optimisation of different security measures: such as *key confidentiality*, *recovery time*, *network size*, *update efficiency*, etc.

In the following subsections we will be working on a specific key update strategy and a specific application profile, that is the *Leave-Based* key update on ZigBee *Home Automation* application profile as a running example to show how we lead to a smart model. Below we list the assumptions for this example:

- **Network size.** The network is initialized to its maximum capacity. TC is not counted as a device in the model, and exists even when there is no device in the network.
- **Join.** A device can join the network if the network capacity is not fully utilized.
- **Leave.** A device can leave the network if there is at least one device other than the TC
- **Compromising leave.** When a device leaves, the key might get compromised with a certain probability
- **Key updating leave.** The counter for leaving devices in initialized to zero, incremented by one in each device leave. Assuming that the threshold is T, right after the leave of the Tth device the key will be updated.

Note that we excluded messaging and key compromise caused by messages for the sake of simplicity.

We first take a classical approach and construct the fundamental model in details in Section 5.3.1. The idea in the classical approach is to represent each device with a unique component. Then we improve the model to be a compact representation of the network such that it will be more scalable and less error-prone in Section 5.3.2. In this phase, we represent the whole network in one single

component. We present the huge difference in number of states and transitions even for moderate instances of classical and compact models. Furthermore, we instantiate minimal instances of the two models and investigate thoroughly to show that not only states and transitions, but also the rates of transitions are one-to-one the same in these models in Section 5.3.3. There we also show that the differences in the number of states and transition are caused by the *duplicates* of the states and transitions in the classical model. In Section 5.3.4, we discuss the bisimilarity of the models, and in Section 5.3.5 we explain how the running example can be extended to support different key update strategies and different network settings.

5.3.1 A Classical Modelling Approach

We start by taking a classical approach that would be a familiar way of modelling such concurrent systems. We consider a realistic model where there is a trust center in the network that regulates key update, and a number of devices that can be either in the network (*joined and authenticated*) or out of the network.

As you can see in Table 5.1, first we start by defining the model type as `ctmc`, i.e. the model will result in a continuous-time Markov chain (CTMC).

CTMCs are frequently used in modelling continuous real time and probabilistic choice. Using CTMCs we can specify the rate of making a transition from one state to another. Probabilistic choice in CTMCs, arises through race conditions when two or more transitions in a state are enabled.

Then we define the constants that we will use throughout the model, which include *rates*, a *probability*, and a *threshold* value. Since this is a CTMC model, obviously we have *rates* for waiting times or delays such as in `join` and `leave` actions. Key compromise has a *probability* therefore we have a way of combining rates with probability. Below we define our constants to be free from any ambiguity.

- A device leaves the network with rate `R_leave`.
- A device joins the network with rate `R_join`.
- A device leaves the network while compromising the key with probability `P_comp`.

The values that we assigned to these constants in Table 5.1 are taken from the ZigBee Home Automation application profile (see details in Appendix B).

```
ctmc

const double R_join   = 1/7;
const double R_leave  = 1/365;
const double P_comp   = 1/100;
const int T_leave;

module TRUSTCENTER
   Comp: bool init false;
   C_leave: [0..T_leave] init 0;

   [joini] true ->  true;
   [leavei] C_leave<T_leave-1 ->   (C_leave'=C_leave+1);
   [leaveCi] C_leave<T_leave-1 ->
                         (Comp'=true) & (C_leave'=C_leave+1);
   [leaveRi] C_leave=T_leave-1 ->(Comp'=false)&(C_leave'=0);
endmodule

module DEVICEi
   Statusi: [0..1] init 1; // 0: outside, 1: inside network

   [joini]  Statusi=0 -> R_join: (Statusi'=1);
   [leavei] Statusi=1 -> R_leave*(1-P_comp):(Statusi'=0);
   [leaveCi] Statusi=1 -> R_leave*P_comp: (Statusi'=0);
   [leaveRi] Statusi=1 -> R_leave: (Statusi'=0);
endmodule
```

Table 5.1: Leave based key update - *classical approach*

Finally, a *threshold* value is required to trigger the key update, that is the threshold `T_leave` for `leave` actions in this strategy.

Next, we have modules representing the entities in real life. As each ZigBee network has a *trust center* to coordinate security-related and administrative activities of the network, we model that with a unique module called `TRUSTCENTER`. The trust center is the authority when a device joins the network, leaves the network, and when the network key should be updated. It keeps a separate counter, `C_leave` that keeps the track of the left devices. When the value of this counter reaches the threshold the network key is updated by trust center. Besides, the trust center is dealing with the key compromise issues by the help of a Boolean variable in the model which reflects the security status of the key. Of course, in real life it is hard to be immediately aware of any key compromise situation, however this is an advantage of modelling, we can see when a key gets compromised depending on our input values such as number of devices, rates,

probabilities, etc.

Before talking about how the entities interact, let us skip to the next module, `DEVICEi`. As a classical modelling attitude in concurrent systems, we need to have the same number of modules as the number of devices. Therefore, the indices, appended as `i` to the end of the module name, local variable, and transitions should actually be substituted by numbers. This actually is a big cumbersome when the number of devices in the module is large. For instance, if we have two devices in the model other than the Trust Center, than we need to have three modules in the model: `TRUSTCENTER`, `DEVICE1`, and `DEVICE2`. However, this is not enough, we need to have unique transitions that would make a device and the trust center interact. Therefore, we need to have unique commands per device, that is for this example `join1`, `join2`, `leave1`, `leave2`, `leaveC1`, `leaveC2`, `leaveR1`, and finally `leaveR2`. Notice that, a device module would only have the relevant commands indicated with proper indices, whereas the Trust Center should have all the commands for that transitions. Also notice that, what we present in Table 5.1 should be unfolded with the proper indices (i.e. having numbers instead of `i`s, and having n number of device modules) to be a proper PRISM source code. As a last note on the device modules, each device is aware of its engagement with the network via the local variable `Status`.

As we started explaining the command labels for transitions, we can now talk about how those entities interact with each other. In real life, joining and leaving the network, also updating the network key, is a pairwise process between a device and the trust center of the network. Similarly, in our model each device is interacting with the trust center over unique transitions. Since we are modelling merely the key update protocols and not the inter-device communication, the devices are supposed to *synchronize* with the trust center, not any other device.

In the model, the modules have actions of `join` type, and `leave` type. The difference between different leave actions is that, a `leave` does not compromise the key, a `leaveC` compromises the key, and a `leaveR` causes the key to be updated (i.e. reset). These actions are fired when the guards of the synchronising modules are satisfied, which are related to the *leave counter*, the *leave threshold*, and the *status of the device*. A very important feature of our modelling is that, each action has a delay with a computed rate. However, the key update action should not be delayed. We achieve this by merging the key update action with the last device leave, which resulted into the action `leaveR`. A more naive approach would be having a separate action for the key update, e.g. `keyupdate` or `reset`, but then it would not be realistic since we would be missing any join, leave or more seriously any key compromise in the delay before the key update and the last leave that causes the key to be updated.

Before advancing to further developments, let us talk about the possible modi-

fications and the criticism on this *classical* model. To have a realistic model, we

```
ctmc

 const int Max;
 const double R_join   = 1/7;
 const double R_leave  = 1/365;
 const double P_comp   = 1/100;
 const int T_leave;

module TRUSTCENTER
   Size: [0..Max] init Max;
   Comp: bool init false;
   C_leave: [0..T_leave] init 0;

   [join]   Size<Max            -> (Max-Size):  (Size'=Size+1);
   [leave]  Size>0   & C_leave<T_leave-1 -> Size:
        (Size'=Size-1) & (C_leave'=C_leave+1);
   [leaveC] Size>0   & C_leave<T_leave-1 -> Size:
       (Size'=Size-1) & (Comp'=true) &(C_leave'=C_leave+1);
   [leaveR] Size>0   & C_leave=T_leave-1 -> Size:
        (Size'=Size-1) & (Comp'=false) &(C_leave'=0);
endmodule

module DEVICE1
   Status1: [0..1] init 1; // 0: outside , 1: inside network

   [join] Status1=0 -> R_join: (Status1'=1);
   [leave] Status1=1 -> R_leave*(1-P_comp):(Status1'=0);
   [leaveC]Status1=1 -> R_leave*P_comp: (Status1'=0);
   [leaveR]Status1=1 -> R_leave: (Status1'=0);
endmodule

module DEVICE2 = DEVICE1 [ Status1=Status2 ] endmodule
system   (DEVICE1 ||| DEVICE2) || TRUSTCENTER      endsystem
```

Table 5.2: Leave based key update - *classical approach revisited*

refrain from having a variable that sets the maximum number of devices in the network, and another variable that keeps the current number of devices that are in the network (notice that being in the *network*, is different than being in the *system*; a device should be engaged by join operations to be *in the network*). In addition, we could have a slightly different model by exploiting the module renaming feature and `system endsystem` construct in PRISM, which allow us to clone modules with a single line of code and have CSP-style parallel composition with process algebra operators, respectively. We present such a modified

model for a network of maximum two devices in Table 5.2, where you can see how we can save from code by cloning `DEVICE1` and then naming the cloned module as `DEVICE2` where all the local variables and actions are automatically renamed by PRISM. In the end of this *revisited* model you can see the synchronisation which works in a way that enables the synchronization between each module and trustcenter (using operator `||`) but not the synchronization within modules (using operator `|||`). This kind of construction is actually mimicing the previous model which achieves synchronisation by distinctive actions that are unique for each device.

Observations on the classical modelling approach:

The classical approach is unfortunately inefficient when the model gets bigger (i.e. having more devices). The required coding is growing as the number of devices grow. The model becomes more and more error-prone since any typo in encoding can risk the sanity of the model. Ironically, the classical approach is much more convincing people in the reliability of the model. That is because, the model is easy to follow, and we are sure that it is doing what we want. In our humble opinion, we can refer to the classical model as modelling in *computer science* perspective.

5.3.2 A Compact Modelling Approach

At this point, we would like to introduce our new modelling approach where we have many improvements in terms of modelling effort and the resulting state space. The nature of the wireless sensor networks leads us to think of a more compact model such that we can have vast number of devices in the network. The classical approach has scalability problems, therefore we need to come up with an idea that is not only easy to model but also produces much smaller state space. We have introduced a naive version of this idea in [YNN$^+$10b], here we explain an improved version. We start by modelling the whole network, i.e. the devices, in a single module. Thus we don't need to have separate modules per device. Then we let this module to be running in parallel with the trust center, similar to the classical approach. The two modules in the model will be synchronising with four different actions, on join, leave and key update purposes. The PRISM code for our model is given in Table 5.3.

As we use more or less the same constants, variables, and actions we refrain from repeating the explanations of those reused components. Instead, we explain what is different in this approach starting with the `TRUSTCENTER` module. We only keep the counter for leaving devices in `TRUSTCENTER`. In other words, Trust Center is merely responsible for implementation and enforcement of the key

```
ctmc

const int Max;
const double R_join   = 1/7;
const double R_leave  = 1/365;
const double P_comp   = 1/100;
const int T_leave;

module TRUSTCENTER
   C_leave: [0..T_leave] init 0;

   [join]   true -> true;
   [leave]  C_leave<T_leave-1 -> (C_leave'=C_leave+1);
   [leaveC] C_leave<T_leave-1 -> (C_leave'=C_leave+1);
   [leaveR] C_leave=T_leave-1 -> (C_leave'=0);
 endmodule

module NETWORK
   Size: [0..Max] init Max;
   Comp: bool init false;

   [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
   [leave] Size>0 -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
   [leaveC] Size>0    -> R_leave*P_comp*Size:
      (Size'=Size-1) & (Comp'=true);
   [leaveR]Size>0->R_leave*Size:(Size'=Size-1)&(Comp'=false);
endmodule
```

Table 5.3: Leave based key update - *new approach*

update strategy. We leave the information keeping for key compromise and network size to the NETWORK module.

The NETWORK module is dealing with the join and leave (including the ones that compromise the key and reset the key) actions. In addition, the status of the key in terms of key compromise is also known by this module. Here we can think of *network size* and *network key* as attributes of the NETWORK module. As we keep the size of the whole network, there is no need to maintain a status variable for device presence as in the classical models.

Our new approach is very simple yet powerful. It can be scaled up and down by just changing the values of the constants, especially the maximum number of devices (Max). There is no need to modify the code unless the key update strategy is changed. Many of the new key update strategies can be integrated by just replacing the trust center module with a new one. Some of the strategies

may of course require modification in network module as well. For example we will show how to implement a precise time-based key update later.

One of the points that should be emphasized again is that key update itself is not a single transition. Modelling key update as a separate action, as it was done in [YNN+10b] would also be fine but causing delay (that is exponentially distributed with a specific rate) on the key update. Instead, we merge the key update action with the `leave` action that causes the key update. Therefore, the model is more realistic (compared to [YNN+10b]) as the update happens instantaneously.

Since we have shown how we save from coding and benefit from easy modelling, now let us come up with a comparison to show how we save from the state space. In Table 5.4, we present number of states and number of transitions for the three models we have explained so far. We instantiate the models for different maximum network sizes and different threshold values, and observe the changes in the number of state and transitions of the CTMC. In the first half of the table, we are considering a network that has maximum two devices at a time, and built the model for threshold values ranging between 1 and 4. We observe that compact model has % 50 less transitions and %25 less states for all threshold values. Obviously, a network of two sensor devices is not realistic for ZigBee networks, therefore we repeat our comparison for more devices. In the second half of the table, we consider maximum 20 devices at a time in the network, and set the threshold value to be 5, 10, 15, and 20. We observe %99.9979973 reduction in the number of states, and %99.9998093 reduction in the number of transitions regardless of the threshold value.

Observations on the compact modelling approach:

The number of states and the number of transitions are extremely reduced in our new and compact approach, and the difference is so big for even a modest network that is up to 20 devices (e.g. some ZigBee application profiles such as WSA and PHHC expect to have around 500 devices in a network, see Appendix B). In fact, the number of states in the classical approach will not be feasible to apply model checking when the number of devices increase. As our aim is to run hundreds of subsequent model checkings (a.k.a. *experiments* in PRISM jargon), the compact model looks very promising with minimum number of states and transitions. The two classical approaches we described above clearly gives the same number of states and numbers for each threshold, as they are two slightly different codings of the same model.

Remember that we have referred to the classical model as a way of modelling in computer science perspective. At this point, we would like to claim that a compact model can be seen as a way of modelling in mathematics perspective.

Table 5.4: CTMC comparison

Size	*Thres.*		*classical*	*classical rev.*	*compact*
2	1	States	4	4	3
		Transitions	8	8	4
2	2	States	12	12	9
		Transitions	28	28	14
2	3	States	20	20	15
		Transitions	48	48	24
2	4	States	28	28	21
		Transitions	68	68	34
20	5	States	9437184	9437184	189
		Transitions	230686720	230686720	440
20	10	States	19922944	19922944	399
		Transitions	492830720	492830720	940
20	15	States	30408704	30408704	609
		Transitions	754974720	754974720	1440
20	20	States	40894464	40894464	819
		Transitions	1017118720	1017118720	1940

Arguably, different disciplines tend to use different models however the results should be the same.

5.3.3 A Closer Look At The Models

After comparing the two approaches in terms of number of states and transitions, we need a more closer look to get more insights. To achieve this, we will instantiate the models for a small number of devices, and examine the resulting CTMCs carefully.

We start by giving a detailed example and showing the relation between those three different models. We instantiate the models for **2 devices** in the network, and enforces leave-based key update with update threshold **2**. In Table 5.5, we present the reachable states for three different models. As you can see, the two classical models which we name as *classical* (and refer to as A), and *classical rev.* (end refer to as B) have the same number of reachable states: 12. However, the

compact model (which we refer to as C) has only 9 reachable states. In Table 5.5, we have also given the values of the variables in the states in parenthesis. The format of the variable ordering is given on the top of each column. For example, we call the first state of model A as $a0$, and showed that in $a0$ state, the values of the variable `Comp`, `C_leave`, `Status1`, and `Status2` are *false*, *0*, *0*, and *0*, respectively.

Table 5.5: Reachable states

	A: *classical* (Comp,C_leave,Status1,Status2)	B: *classical rev.* (Size,Comp,C_leave,Status1,Status2)	C: *compact* (Size,Comp,C_leave)
0	a0:(false,0,0,0)	b0:(0,false,0,0,0)	c0:(0,false,0)
1	a1:(false,0,0,1)	b1:(0,false,1,0,0)	c1:(0,false,1)
2	a2:(false,0,1,0)	b2:(0,true,1,0,0)	c2:(0,true,1)
3	a3:(false,0,1,1)	b3:(1,false,0,0,1)	c3:(1,false,0)
4	a4:(false,1,0,0)	b4:(1,false,0,1,0)	c4:(1,false,1)
5	a5:(false,1,0,1)	b5:(1,false,1,0,1)	c5:(1,true,1)
6	a6:(false,1,1,0)	b6:(1,false,1,1,0)	c6:(2,false,0)
7	a7:(false,1,1,1)	b7:(1,true,1,0,1)	c7:(2,false,1)
8	a8:(true,1,0,0)	b8:(1,true,1,1,0)	c8:(2,true,1)
9	a9:(true,1,0,1)	b9:(2,false,0,1,1)	
10	a10:(true,1,1,0)	b10:(2,false,1,1,1)	
11	a11:(true,1,1,1)	b11:(2,true,1,1,1)	

An important result of Table 5.5 is all the states in the classical model also exist in the revised classical model decorated with a *Size* prefix varying between 0, 1, and 2. In fact, we can precisely point which state in model A is similar to which state in model B. That would actually result in an *injective* (or *one-to-one*) mapping function, as we present in Table 5.6. These two models, A and B, also has a relation to model C. However, since model C has less reachable states, as an analogy to functions, the relation from A (and B) to C is not injective but *surjective* (or *onto*). For example, the states $c3$, $c4$, and $c5$ correspond to more than one states in models A and B.

As we have presented the reachable states, now we can continue with the transitions in the models. In Table 5.7, we listed all the transition in three models. There exists a separate column for each model, and within each column we have follow a format where each row is a unique transition that we write as a triple: *origin state*, *destination state*, and *rate*. Note that we omitted the prefixes (i.e. A, B, or C) in the state numbers to save space. As an example, the first line

Table 5.6: Similarity between the states

A: *classical*	B: *classical rev.*	C: *compact*
a0	b0	c0
a1	b3	c3
a2	b4	c3
a3	b9	c6
a4	b1	c1
a5	b5	c4
a6	b6	c4
a7	b10	c7
a8	b2	c2
a9	b7	c5
a10	b8	c5
a11	b11	c8

of the last column tells us that there is a transition in model C that is from state $c0$ to state $c3$ with a rate of 0.285714285714. As in the sates, we have less transitions in model C, that is only 14. The models A and B has 28 transitions for each.

As shown in Table 5.7, model B includes all the transitions in model A, even though there is no `Size` variable in A. This fact does not mean that we can remove the `Size` variable from model B, because we use it to limit oversize and undersize (leave and join) actions in the network. Besides, the order of transitions are not the same in A and B, just because of the different state numbering in the reachable states (see Table 5.5). To help the reader to follow the example, we list the similarity in the transitions in Table 5.8. Note that, in order to distinguish between the states and the transitions we use different prefixes in Table 5.8, i.e. TrA, TrB, and TrC.

The similarity between C and the other model is now even more interesting since there are two transitions in A and B for each transition in C. Thus, we can say that classical models include all the transitions in our compact model and all of them are duplicated. The reason of the duplicates is the number of devices, or more technically the local variables `Status 1` and `Status 2` duplicating each transition in model C.

The Tables 5.7 and 5.8 show that for the same transitions, all three models have

Table 5.7: Transitions

	A: *classical*	B: *classical rev.*	C: *compact*
1	0 1 0.142857142857	0 3 0.285714285714	0 3 0.285714285714
2	0 2 0.142857142857	0 4 0.285714285714	1 4 0.285714285714
3	1 3 0.142857142857	1 5 0.285714285714	2 5 0.285714285714
4	1 4 0.00271232876712	1 6 0.285714285714	3 1 0.00271232876712
5	1 8 2.7397260274e-05	2 7 0.285714285714	3 2 2.7397260274e-05
6	2 3 0.142857142857	2 8 0.285714285714	3 6 0.142857142857
7	2 4 0.00271232876712	3 1 0.00271232876712	4 0 0.0027397260274
8	2 8 2.7397260274e-05	3 2 2.7397260274e-05	4 7 0.142857142857
9	3 5 0.00271232876712	3 9 0.142857142857	5 0 0.0027397260274
10	3 6 0.00271232876712	4 1 0.00271232876712	5 8 0.142857142857
11	3 9 2.7397260274e-05	4 2 2.7397260274e-05	6 4 0.00542465753425
12	3 10 2.7397260274e-05	4 9 0.142857142857	6 5 5.47945205479e-05
13	4 5 0.142857142857	5 0 0.0027397260274	7 3 0.00547945205479
14	4 6 0.142857142857	5 10 0.142857142857	8 3 0.00547945205479
15	5 0 0.0027397260274	6 0 0.0027397260274	
16	5 7 0.142857142857	6 10 0.142857142857	
17	6 0 0.0027397260274	7 0 0.0027397260274	
18	6 7 0.142857142857	7 11 0.142857142857	
19	7 1 0.0027397260274	8 0 0.0027397260274	
20	7 2 0.0027397260274	8 11 0.142857142857	
21	8 9 0.142857142857	9 5 0.00542465753425	
22	8 10 0.142857142857	9 6 0.00542465753425	
23	9 0 0.0027397260274	9 7 5.47945205479e-05	
24	9 11 0.142857142857	9 8 5.47945205479e-05	
25	10 0 0.0027397260274	10 3 0.00547945205479	
26	10 11 0.142857142857	10 4 0.00547945205479	
27	11 1 0.0027397260274	11 3 0.00547945205479	
28	11 2 0.0027397260274	11 4 0.00547945205479	

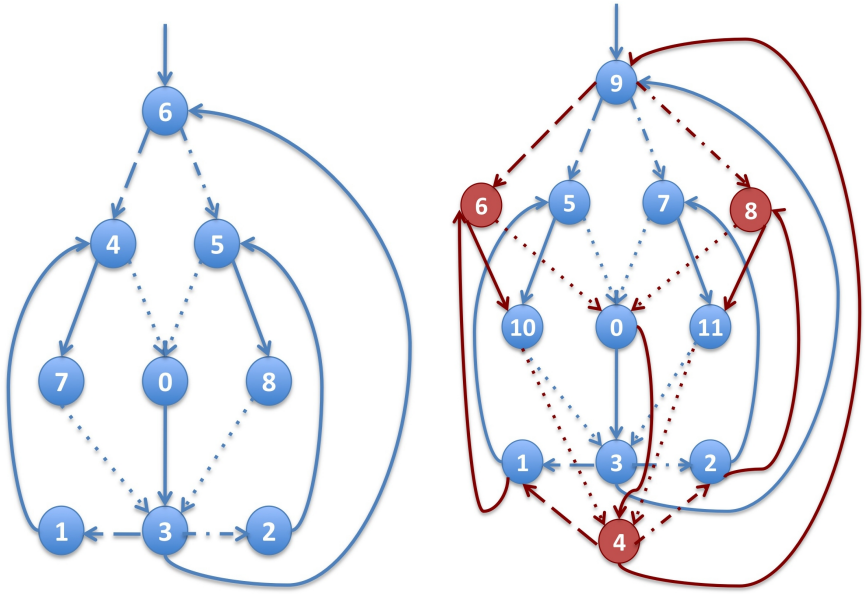
the same rates.

It is now easier to match the transitions in CTMCs with our corresponding PRISM models. For example the transitions $c7$, $c9$, $c13$, and $c14$ all correspond to the `leaveR` (a.k.a reset or update) action in the PRISM model. The reason of having four different transitions is obviously the guard of the command can be satisfied in four different ways depending on the values of the variables in the guard.

Using the results in Table 5.5 and Table 5.6, we can show that models A and B (i.e. classical and classical revised models) actually have the same states. Using the results in Table 5.7 and Table 5.8, we can show that models A and B actually have the same transitions with exactly the same rates. Now, using these four tables we can conclude that the models A and B produce exactly the same continuous-time Markov Chain for the instantiation of maximum 2 devices and leave threshold of 2. From now on, we will continue with only two models: B *classical rev.*, and C *compact*.

Table 5.8: Similarity between the transitions

A: *classical*	B: *classical rev.*	C: *compact*	A: *classical*	B: *classical rev.*	C: *compact*
TrA1	TrB1	TrC1	TrA15	TrB13	TrC7
TrA2	TrB2	TrC1	TrA16	TrB14	TrC8
TrA3	TrB9	TrC6	TrA17	TrB15	TrC7
TrA4	TrB7	TrC4	TrA18	TrB16	TrC8
TrA5	TrB8	TrC5	TrA19	TrB25	TrC13
TrA6	TrB12	TrC6	TrA20	TrB26	TrC13
TrA7	TrB10	TrC4	TrA21	TrB5	TrC3
TrA8	TrB11	TrC5	TrA22	TrB6	TrC3
TrA9	TrB21	TrC11	TrA23	TrB17	TrC9
TrA10	TrB22	TrC11	TrA24	TrB18	TrC10
TrA11	TrB23	TrC12	TrA25	TrB19	TrC9
TrA12	TrB24	TrC12	TrA26	TrB20	TrC10
TrA13	TrB3	TrC2	TrA27	TrB27	TrC14
TrA14	TrB4	TrC2	TrA28	TrB28	TrC14



Compact model (C) *Classical (revised) model* (B)

Figure 5.1: The state-transition diagrams

At this point, our aim is to show that the compact model is not only efficient, scalable, and compact but also behaving no different than the classical model. In Fig. 5.1, we present the state-transition diagrams of the models. Instead of having labels on the transitions, we classify them with line patterns.

- A *dashed* line is a `leave` action
- A *dashed-and-dotted* line is a `leaveC` action
- A *dotted* line is a `leaveR` action
- A *solid* line is a `join` action

In the classical model, we also have color coding such that the duplicated states and transitions are in red. Since there are two devices in the network, whenever an action (variants of `leave`, or `join`) is fired, it might either be *device1* or *device2*. These two combinations cause the red states and transitions.

So if we continue our prefixed fashion of naming the states, we can say that

- states $(b2, b7, b8, b11)$ and $(c2, c5, c8)$ are where the key is compromised,
- states $(b0, b3, b4, b9)$ and $(c0, c3, c6)$ are where the number of leaving devices in the network is (set as) 0,
- states $(b0, b1, b2)$ and $(c0, c1, c2)$ are where the number of devices in the network is 0,
- states $(b0, b1, b2)$ and $(c3, c4, c5)$ are where the number of devices in the network is 1,
- states $(b9, b10, b11)$ and $(c6, c7, c8)$ are where the number of devices in the network is 2.

In conclusion, we can say that the states $c3$, $c4$, and $c5$ are duplicated as ($b3$, $b4$), ($b5$, $b6$), and ($b7$, $b8$), respectively. Besides the transitions from and to all these states are also exactly duplicated. We show for this minimal model and claim that compact model is modelling the same system as classical model but with less states and less transitions.

All the statements above are verifiable using the states in Table 5.5. Note that, the figures are CTMCs where the self loops and rates are omitted for the sake of simplicity.

5.3.4 A Discussion on Bisimilarity of the Models

As we have explained in Section 4.5 of the preliminaries chapter, *bisimulation* is a binary relation between state-transition systems, that associates systems which behave in the same way. Intuitively two systems are *bisimulation-equivalent* or *bisimilar* if they match each other's moves. In this sense, the systems cannot be distinguished from each other by an observer.

Although in the previous section we have shown that for a network of maximum two devices, and a key update threshold of two leaving devices, our two different models have a huge similarity in terms of states, transitions, and rates; bisimulation can help us in verifying if these two models can simulate each other in a more theory-driven way.

In this section, we present a small discussion on the bisimilarity of our models until now. We will first make use of the definition of bisimulation equivalence as a relation between two transition systems in Section 4.5. The equivalence of the compact model and the classical model for two devices and threshold of two follows the fact that the relation

$$\begin{aligned}\mathcal{R} \quad = \quad & \{(c6, b9), (c4, b5), (c4, b6), (c5, b7), (c5, b8), (c7, b10), \\ & (c0, b0), (c8, b11), (c3, b3), (c3, b4), (c1, b1), (c2, b2)\}\end{aligned}$$

is a bisimulation, which can easily be verified on Fig. 5.1.

An alternative perspective on bisimulation is to consider bisimulation relation between states in a single transition system, rather than a relation between transition systems. This sense of bisimulation can be used in obtaining smaller models out of large models. Thus, this time we will make use of the definition of bisimulation equivalence as a relation on states in Section 4.5.

To show bisimilarity, first we will decorate the transition systems in Fig. 5.1 with the rates. Since we continue with our two simple models we will take the rates from Table 5.7. Remember that the two transition matrices had exactly the same rates for corresponding transitions. In Table 5.9, we label the rates (not the actions) in order to increase readability of the transition diagram. Thus, we lead to Fig. 5.2 where each transition has a *rate* that is marked with a letter, and an action that causes the transition that is shown with a line pattern. As an example, taking a look at the compact model in Fig. 5.2 we can say that state 2 and state 3 (or C2 and C3, to be in accordance with Table 5.5) both have a transition that is shown with a solid line, i.e. a `join` action. However these two solid lines have different labels on them, **a** and **d**, meaning that the rates of those transitions are different and can be looked up in Table 5.9.

Table 5.9: Labeling transitions

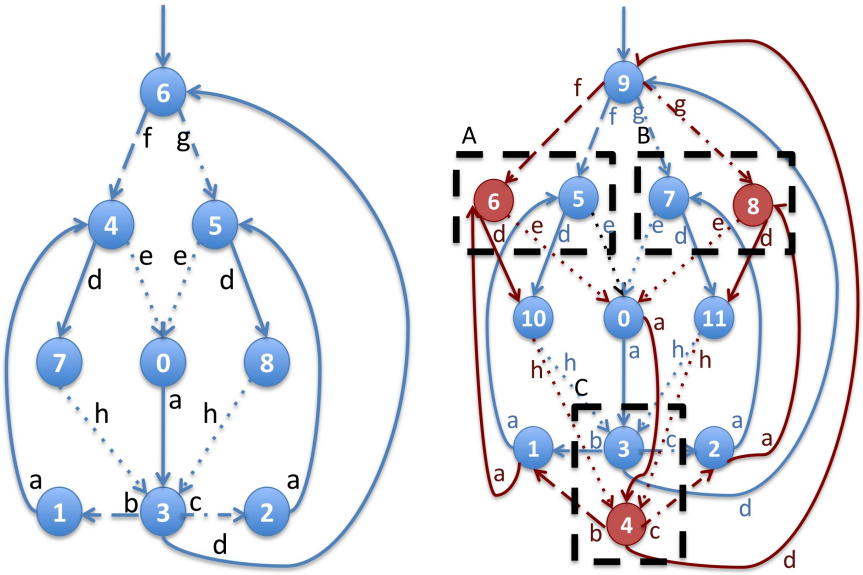
Label	**Rate**	**Computation**
a	0.285714285714	`R_join` * `Size`, `Size`=2
b	0.00271232876712	`R_leave` * (1-`P_comp`) * `Size`, `Size`=1
c	2.7397260274e-05	`R_leave` * `P_comp` * `Size`, `Size`=1
d	0.142857142857	`R_join` * `Size`, `Size`=1
e	0.0027397260274	`R_leave` * `Size`, `Size`=1
f	0.00542465753425	`R_leave` * (1-`P_comp`) * `Size`, `Size`=2
g	5.47945205479e-05	`R_leave` * `P_comp` * `Size`, `Size`=2
h	0.00547945205479	`R_leave` * `Size`, `Size`=2

As we have our finite continuous-time Markov chain now we can define an equivalence relation $\approx$ on our state set (S) such that for any equivalence class A, it does not matter which state within the equivalence class we are in. When we *run* the Markov chain, we only care about the sequence of the equivalence classes entered. Then, as explained in [LS91], we can form a quotient Markov chain S/$\approx$ whose states are the equivalence classes of $\approx$, if and only if $\approx$ is a probabilistic bisimulation. Namely, the probability of going (in one step) from a state a in a class A to some state b in a class B is independent of a:

$$\sum_{b \in B} p_{ab} = \sum_{b \in B} p_{a'b}$$

where p_{ab} is the transition probability from state s to state t.

As shown in Fig. 5.2, we formed equivalence classes of $\approx$, that is A, B, and C denoted by dashed boxes. You can consider the rest of the states as unique equivalence classes, for instance $0 \in D$, $1 \in E$, $2 \in F$, $9 \in G$, $10 \in H$, and $11 \in I$. In this case, $\approx$ is a probabilistic bisimulation because all the states in the same equivalence class have the same rates of going in one step to another equivalence class. For instance, state 6 from class A, can go to state 10 (class H) and 0 f(class D) with rates d, and e, respectively. The same rates hold for state 7, which is the other state in the same source class (A) when going to the same destination classes (D and H). Besides, the incoming transitions to states 5 and 6 have the same rate, f, from the class G.



Compact model (C) *Classical (revised) model* (B)

Figure 5.2: The state-transition diagrams, revisited

5.3.5 Extending Model with Different Key Update Strategies

Up to now, we developed our model for the running example of the leave-based key update on the ZigBee Home Automation application profile. However, different key update strategies (and of course differently configured models to fit various application profiles) can be implemented and those should be verified and analysed using stochastic model checking as well. In fact, the mere key update strategy that is suggested in the ZigBee specification is what we name as the *time-based* strategy, and the leave-based one is actually our proposal.

Time-based key update. We start modelling the time-based key update similar to the model in [YNN+10b], presenting the result in Table 5.10. You will notice that the difference with the leave-based (compact) model is very limited. Instead of `T_leave`, we have `T_time` that corresponds to the threshold in time-based model (also known as the *key expire time*). This threshold is actually an integer constant, the number of *months* that the key is valid. An important difference is, instead of a type of leave action that causes the update we model a `reset` action whose only purpose is the key update. We define the rate of the `reset` action as `R_reset` and define it as `1/(30*T_time)`, note that

we designed our time unit to be a *day* therefore a period needs to be multiplied by 30 to be converted as a month.

```
ctmc

const int Max;
const double R_join   = 1/7;
const double R_leave  = 1/365;
const double P_comp   = 1/100;
const int T_time;
const double R_reset  = 1/(30*T_time);

module TRUSTCENTER
   [join]   true -> true;
   [leave]  true -> true;
   [leaveC] true -> true;
   [reset]  true -> R_reset: true;
 endmodule

module NETWORK
   Size: [0..Max] init Max;
   Comp: bool init false;

   [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
   [leave]  Size>0 -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
   [leaveC] Size>0    -> R_leave*P_comp*Size:
      (Size'=Size-1) & (Comp'=true);
   [reset] true->1:(Comp'=false);
endmodule
```

Table 5.10: Time based key update - *a naive approach*

The time-based key update model in Table 5.10 is also very compact, and very effective. However, as a CTMC model the key update period is exponentially distributed with a rate of `R_reset` (more precisely, the product of `R_reset` from module `TRUSTCENTER` and `1` from module `NETWORK`. This kind of modelling does not cause a burden on computing steady-state probabilities, but it is not very accurate indeed. In other words, the key updates does not exactly happen every `T_time` months.

In a sense, a *periodical* event is not fitting a continuous-time Markov chain. However, event such as join and leave are perfectly modelled with that. We have to add deterministic time delays to the CTMC model. At this point we try to approximate deterministic delays with *phase-type* distribution. We implement Erlang distribution, a special case of a phase-type distribution, that

is the distribution of the sum of k independent identically distributed random variables each having an exponential distribution. The rate of the Erlang distribution is the rate of this exponential distribution. By doing that, we are able to continue with our CTMC model that we developed up to now, and also use PRISM to compute probabilistic model checking results.

Erlang distribution has two parameters, a non-negative integer *shape* parameter k and a non-negative real number *rate* parameter λ. When the shape parameter k equals 1, the distribution simplifies to the exponential distribution. In Table 5.11, we present the modifications to convert our naive model to the improved time-based key update model. Notice that, `NETWORK` module remains the same and `R_reset` is no more needed. In order to implement the Erlang distribution we introduced the shape and the reciprocal of the rate ($mean = 1/\lambda$) parameters as constants `k` and `mean`, respectively. We compute the sum of `k` independent identically distributed random variables each having an exponential distribution in the `TRUSTCENTER` using the local counter `i` and a transition that does not synchronise with the `NETWORK` module.

```
const int k;
const double mean = 30*M;

module TRUSTCENTER
   i : [1..k+1];

   [join]   true -> true;
   [leave]  true -> true;
   [leaveC] true -> true;
   []       i<k -> k/mean :   (i'=i+1);
   [reset]  i = k -> k/mean : (i'=1);
 endmodule
```

Table 5.11: Time based key update - *using phase-type distribution*

The improved version of the time-based key update is more accurate on the time of key updates, however quite costly. Increasing the value of the shape parameter gives us more accurate results in modelling deterministic time delays i.e. the key update periods. However, there is a trade-off here between the accuracy and the size (state space) of the model. Roughly, the model size is increased by a factor of k.

Join-based key update. This key update strategy is very similar to the leave-based key update in the implementation. Technically, the key update will be triggered when a certain number (i.e. number of update threshold) of devices joined the network. Since the changes in the model are trivial we don't list the

model details here but in Appendix C, as we did for all other methods.

Message-based key update. Here we mention a new key update method that was proposed by [MAK08], which is built on the idea that the messages communicated over the network are actually increasing the risk of an attack. We will refer to this method as *message-based* key update (MB) throughout this thesis. This approach counts the number of communicated messages and issues a key update after a certain number of messages communicated.

Implementing various application profiles. As we have explained before, different application profiles can mainly be characterized by their join/leave rates, key compromise probabilities, and maximum network size. To embody this information we don't need to make significant changes in the models. All we need to do is to initialize the constants `Max`, `R_join`, `R_leave`, and `P_comp` with proper values.

Implementing costs and rewards. As we have explained in Chapter 4, the probabilistic model checker we use gives the possibilities of using multiple reward structure in a model, and regards costs also as rewards. We make use of this and specify both state and transition rewards in our models. We investigate cumulative, instantaneous, and steady-state reward properties. By doing so, we can compute expected total time that a compromised key was used, expected number of key compromise cases, expected number of key updates in each strategy, expected number of useful and useless key updates, and more. Therefore, reward structures are vey useful in the verification of security and security-related protocols. We present the usage of reward structures and relevant property specification in the following sections. Besides, present the listing of the reward structures and logic formulae in Appendix C.

Chapter 6

Analysing Scenarios

In this chapter, we try to explain how stochastic model checking technique can be applied to a part of a wireless network standard's security sublayer such that we can reason about security properties and get quantitative answers to our questions. We present a methodology for determining optimal key update policies and security parameters, considering the security needs for realistic application scenarios.

Our analysis is in three key aspects: *confidentiality*, *recovery*, and *efficiency*. In Section 6.1, we introduce our analysis in key confidentiality both in the long run and in single time instants. In Section 6.2.1, we introduce our analysis in recovery of a key compromise such that the mean time to recover would be minimized and maximum recovery period would be limited. Finally, in Section 6.3 we introduce our analysis in efficiency which considers both costs and usefulness of the updates.

Our running example throughout this chapter is: leave-based key update versus time-based key update. Thus we also show that stochastic model checking can be efficiently used in determining the most appropriate key update strategy for an intended security level. What we mean by *key update strategy* is a defined key update method with a defined update threshold value, and we stick to this definition throughout the chapter.

6.1 Optimising Key Confidentiality

Confidentiality of communication over a network where cryptography is employed is strictly bound to the confidentiality of the keys that are used for encryption and decryption, as stated in Kerckhoffs's principle. Therefore, in this section we are actually optimising the confidentiality of the network communication by optimising the confidentiality of the network key.

Using our models with proper configurations (either using the constants that we derived for specific ZigBee application profiles in Appendix B, or using custom values for specific network designs), we can obtain the probability of being in a state where the key is compromised. We can compute this probability not only for the long run, but also for a specific time instant.

6.1.1 Risk of Key Compromise in the Long Run

One of the fundamental concerns in security is the preservation of confidentiality in the *long-run*. Given the properties of a network such as estimated risk of each device and the environment, we can compute the steady-state probability of the occurrence of an event. Thus we are able to answer the question:

What is the probability of the key being compromised in the long run?

Using stochastic model checking, we can obtain the steady-state probability of key compromise for different key update methods and different key update thresholds. Below is the continuous stochastic logic (CSL) [ASSB96] formula that formally specifies the question above in PRISM notation:

```
S=?  [ Comp ]
```

The `S` operator is used for reasoning about the *steady-state* probability of the occurrence of an event. In this case, we are asking what is the probability of `Comp` being true in the long run (or *equilibrium*). Since `Comp` is the Boolean variable that is true when the key is compromised, we actually ask the probability of the key being compromised.

As an example, we present the results for the *Home Automation* (HA) application profile, using the *time-based* (**TB**) and *leave-based* (**LB**) key update

strategies in Fig. 6.1. The x-axis shows the values of the thresholds of the key update strategies: `M` denotes the threshold for TB (short form of `T_time`) and `N` denotes the threshold for LB (short form of `T_leave`). On the y-axis we have the steady-state probability (or risk) that the key is compromised. The curves represent the results for the two key update strategies, such that each point gives the steady-state probability of the key being compromised when that key update method is implemented with the corresponding threshold value in the x-axis.

Since the experiments shown here are for the Home Automation application profile of ZigBee standard, the parameter values are: `R_join`=1/7, `R_leave`=1/365, `P_comp`=1/100, `Max`=20. For the sake of simplicity, we consider the models that the key gets compromised by only compromising leave actions, but not by the communication over the network. The time unit is taken as 1 day, and phase-type distribution is employed for the deterministic delays in TB such that the shape parameter k for the probability distribution is taken as 1000.

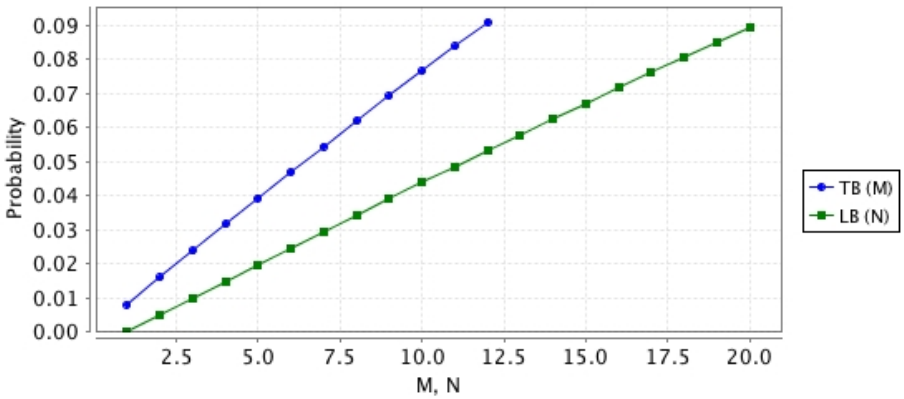


Figure 6.1: Probability of key compromise in the long run in *Home Automation* using a) *time-based* (threshold M) b) *leave-based* (threshold N) key update.

Fig. 6.1 tells us that, if we use TB strategy picking a threshold between 1 and 12 (months) then the risk of key compromise will be between 0.8% and 9% for the specified application profile above. Similarly, if we use LB strategy picking a threshold between 1 and 20 (leaving devices) then the risk of key compromise will be between 0% and 8.9%. As a more concrete example, if we update the key every three months in HA application profile, then we expect average key compromise risk to be 2.38% according to the results. On the other hand, if we update the key after every 5th device leaving the network, then the risk will be 1.98%.

At this point we would like to mention that, if we model the network to face key compromise only by leaving devices, then updating the key after each leave – i.e. LB with threshold value 1 – will result in zero risk which is also observable in Fig. 6.1. We would also like to point out that even though the results seem to produce two *line*s in reality they are not. The results graphically exhibit a concave function's behaviour. In other words, the results have a decreasing slope, i.e. the derivative of the function is decreasing. Practically speaking, the difference between the steady-state probabilities when the threshold is `T` and `T+1` is getting smaller as `T` value grows.

The results also reveal which strategy can be substituted for another. In other words, we can conclude that, for example, TB with a threshold of m is equivalent/substitutable with, for example, LB with a threshold of n in terms of key compromise probability in the long run.

In terms of comparison, steady-state probabilities are very useful since they allow us to find *comparable* strategies. This way we can refine the strategies to be analysed, and save time and memory.

Benefiting from this reasoning, we are also able to answer the question:

> *What is the optimum key update threshold value I should set, if I am able to enforce X key update strategy and I would like to have the probability of the key being compromised in the long run less than Y percent?*

As such, it is very useful to know the risk in the long run when comparing different strategies and different threshold values. We will use this property in our case studies in the following chapter.

6.1.2 Maximum Risk of Key Compromise

Estimating the risk of key compromise is no doubt important when optimising the key confidentiality. However, the risk is not the same at each time instant and surely we would like to know more about the maximum risk that we could face, and the pattern of the encountered risk over time.

As a starting point, we would like to be able to answer questions such as:

> *What is the probability that the key is compromised 6 months from now?*

Obviously the answer depends on how often we are changing the key and we shall therefore pose the questions for different replacement strategies. Below is the CSL formula that formally specifies the question above in PRISM notation:

```
P=?  [ F[30*T,30*T] Comp ]
```

The `P` operator is used for reasoning about the probability of the occurrence of an event, and in this case it is the *transient* probability that we are interested in. We are asking what is the probability of `Comp` eventually being true at time instant `30*T`. Since `Comp` is the Boolean variable that is true when the key is compromised and assuming that the time unit is a day in our models, the formula can be translated as "*what is the probability of the key being compromised* `T` *months after the network starts operation?*".

Continuing with our previous example, we present the results again for the *Home Automation* scenario, using TB and LB key update strategies but this time for the transient probability in Fig. 6.2. The x-axis shows the month `T` of interest and spans two years period. On the y-axis we have the probability that the key is compromised. The curves represent different thresholds (i.e. M and N) for resetting the key such that each point gives the probability of the key being compromised at that time instant

The dotted lines represent the leave-based key update strategy for different threshold values (i.e. update takes place after in total of N devices left the network). The solid lines represent the time-based key update strategy for different period values (i.e. update takes place every M months).

For the TB key update strategy where the key is updated at fixed time intervals, we analyse scenarios of resetting the key every 3 months, every 6 months, etc. up to every 12 months. As an example we see that if we replace the key once every 6 months then the green curve (in fact not a curve but a sawtooth form) shows that the probability of the key being compromised 6 months from now is 4.8%; if we replace it every 12 months then the corresponding probability is 9.2%.

For the LB key update strategy where the the key is changed whenever a fixed number of devices has left the network, the curves show the results for threshold of 5, 10, 15 and 20 devices. As an example we see that if we replace the key whenever 10 devices have left the network then the green curve shows that the probability of the key being compromised after 6 months is 4.5%; if we replace it after 20 devices have left the network then the corresponding probability is 9.2%.

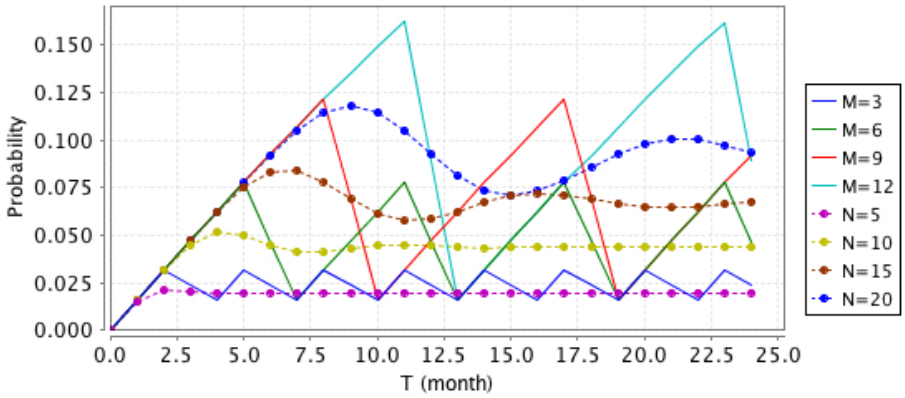


Figure 6.2: Transient probability of key compromise in *Home Automation* using a) *time-based* (threshold M) b) *leave-based* (threshold N) key update.

The results show us that the leave-based key update strategy provides a fairly stable risk for especially low threshold values. Besides we learn that in terms of the maximum key compromise probability, the two key update strategies are comparable for (M=9,N=20), (M=6,N=15), and (roughly) (M=3,N=5).

Benefiting from this reasoning, we are also able to answer the question:

> *What is the optimum key update strategy I should enforce, if the maximum key compromise probability we can tolerate is less than X percent at any time?*

Besides it is very useful to know the pattern of the instant risk before employing a key update strategy. Some networks may tolerate instant high risks for the sake of relatively low risk in the long run, whereas some could prefer a more stabile risk pattern. We present more details on the fluctuations in the results of LB (and also other similar key update methods) in Appendix D.2.

6.1.3 Methodology for Optimising Key Confidentiality

In the previous two sections, we have explained how to analyse key confidentiality from two different points of view. In this section, we present our methodology for determining optimum key update strategy concerning key confidentiality.

In Table 6.1, we present the methodology as an algorithm of six steps. The idea is to find a solution satisfying networks key confidentiality requirements. The solution is computed in two sequential steps, making use of the developments we have explained above. Since the methodology in Table 6.1 is almost self-contained, we would like to express additional points in the methodology. First point is the observation details in transient analysis, which could be defined in STEP 1.b and used in STEP-V. The observation period and observation frequency should be specified for transient analysis. The second point is the solutions returned by the method. Obviously, the method can return empty set, or multiple results. In any of these cases, the requirements maybe refined and/or reset in order to achieve a better solution.

Surely the results on Fig. 6.1 and Fig. 6.2 shed some light on how the key update strategy influences the risk of key compromise. But offhand they do not really give exact answers to our questions. So let us try to ask a more direct question, and make use of both of the previous results:

> *What is the best replacement strategy if it the acceptable probability of key compromise in the long run should be between 5% and 9%, and acceptable probability of maximum key compromise at any time should be between 8.5% and 10%?*

In this case, the question sets the requirements clearly. We can start by applying the first step in our methodology such that

$$a_{min} = 0.05, \quad a_{max} = 0.09, \quad b_{min} = 0.085, \text{ and } b_{max} = 0.1$$

Then, complying with the previous examples, we can set our capabilities in STEP–II as

$$KU = \{TB, LB\}$$
$$TS = \{TS_{TB}, TS_{LB}\}$$
$$TS_{TB} = \{TS_{TB}[1], \ldots, TS_{TB}[12]\}$$
$$TS_{LB} = \{TS_{LB}[1], \ldots, TS_{LB}[20]\}$$

STEP I: Set the requirements
1.a) average risk should be between a_{min} and a_{max}
1.b) maximum risk should be between b_{min} and b_{max}

STEP II: Set the capabilities
2.a) the set of key update methods that can be implemented:
$KU = \{KU_1, \ldots, KU_n\}$
2.b) the multiset of threshold sets that includes
an implementable threshold set composed of positive
integers for each key update method in KU:
$TS = \{TS_1, \ldots, TS_n\}$

STEP III: Initialize the solution set as:
$SS = \emptyset$

STEP IV: Compute the solution set that satisfies 1.a

For all i in KU do
For all j in TS do
For all k in TS_j do
1. compute the steady-state prob. $S_{TS_j[k]}[Comp]$
2. if $S_{TS_j[k]}[Comp] \models$ 1.a then
$SS = SS \cup TS_j[k]$

STEP V: Refine the solution set such that it also satisfies 1.b

For all i in SS do
1. compute the transient prob. $P_{SS[i]}[F_t\ Comp]$
2. if $P_{SS[i]}[F_t\ Comp] \not\models$ 1.b then
$SS = SS \setminus SS[i]$

STEP VI: Return the solution set SS.

Table 6.1: Methodology for optimising key confidentiality

Starting with an empty solution set, we need to compute the solution that satisfies the requirement in STEP–I.a, as explained in STEP–IV of the methodology. Notice that we can safely reuse the results in Fig. 6.1 and compute the satisfying threshold values as $\{7, 8, 9, 10, 11\}$ for TB and $\{12, 13, 14, 15, 16, 17, 18, 19, 20\}$ for LB. Thus the solution set has 14 elements, the union of the satisfying strategies.

In the next step, we need to refine the solution set by removing the threshold values that do not satisfy the requirement in STEP–I.b. This time we can

make use of the results in Fig. 6.2 and find out that in TB threshold values $\texttt{M} < 6$ and $\texttt{M} > 9$, in LB threshold values $\texttt{N} < 15$ and $\texttt{N} > 20$ does not satisfy the requirements in STEP–I.b. This actually reduces our effort for STEP–V, namely we can apply model checking for a refined set of threshold values:

$$TS_{TB}[7], TS_{TB}[8], TS_{LB}[16], TS_{LB}[17], TS_{LB}[18], TS_{LB}[19]$$

To make another refinement before computing the solutions, we should choose the maximum `T` value such that large enough to cover largest periodic oscillation (to be able to see the maximum probability), and small enough to complete the model checking in a reasonable time (and memory). The clue we get is, instead of 24 months we can run the checking for less than 16 months. This will save us 54 (9 `T` values times 6 threshold values) which actually amounts to 37% of the model checkings.

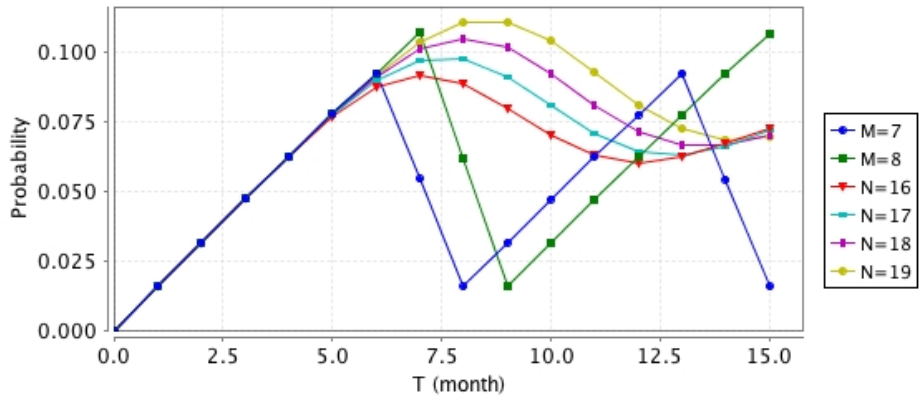


Figure 6.3: Probability of key compromise in *Home Automation* using a) *time-based* (threshold M) b) *leave-based* (threshold N) key update - *focusing around 10% key compromise.*

We present the results for the set of threshold values in two update strategies that will lead to the answer we are looking for in Fig. 6.3. Indeed the graphs give us guidelines here: If we replace the key at regular intervals then we should do it (at least) every 7 months. Alternatively, we could replace the key whenever (at most) 17 devices have left the network.

But which of these two strategies should we prefer? How can we compare them? These questions wait to be answered in the following sections.

6.2 Optimising Recovery From Key Compromise

In networking area, the term *key recovery* is often used for transportation of the valid key to a device which somehow lost its key. In wireless sensor networks, a failure in the battery could easily require a key recovery in this sense. However, we would like to define another type of *key recovery* that we will use in this thesis. Briefly, we define the action of issuing a key update when the network key was compromised as recovering the key. In this section, we define two measures to optimise key recovery.

6.2.1 Mean Time to Recover

A security key may get compromised for several reasons, and eventually it will be updated (or reset). However, the time needed to recover from a compromise case needs to be optimized in such a way that the network is not without protection for a long time period. Thus, we are interested in the (expected) *mean time to recover* (**MTTR**) from a compromise of the key. Rather than formally defining it, we will illustrate by an example in Fig. 6.4.

Assume that we design a ZigBee network, with a number of devices and each device in the network has a valid network key for secure communication. Independent of the key update strategy, we can safely assume that the network key will be updated after some time. In fig. 6.4, we label the starting time of the network as `start` and a solid line following the start corresponds to the time where the network key is safe, i.e. not compromised. Then, we observe a thick cross on the solid line that represents a key compromise, we label this time point as $\mathtt{C}_1$. After point $\mathtt{C}_1$, the line is no more solid but dotted which means that the valid network key is compromised, therefore the communication might not be secure anymore. Note that any key comprising event won't make any changes because the key is already compromised. After some triggering event depending on the type of the key update strategy (e.g. a fixed amount of time for the time-based strategy, or a fixed number of leaving devices for the leave-based strategy) we will have a key update event. The first key update event in our little example in Fig. 6.4 is labelled as `update` and also $\mathtt{KU}_1$. Here it is necessary to mention that in the figure we don't have two different time lines, we just wanted to make the figure more readable by duplicating the timelines. At the point $\mathtt{KU}_1$, the compromised key is *revoked* and devices start using a new and *fresh* network key. As we have labelled as $\mathtt{R}_1$ in Fig. 6.4, we refer to the time period between the time point that a fresh key become compromised, and the time point that it is recovered by a key update a *recovery* time. In the example, we have two key updates $\mathtt{KU}_1$ and $\mathtt{KU}_2$ which are followed by two recovery periods

R_1 and R_2.

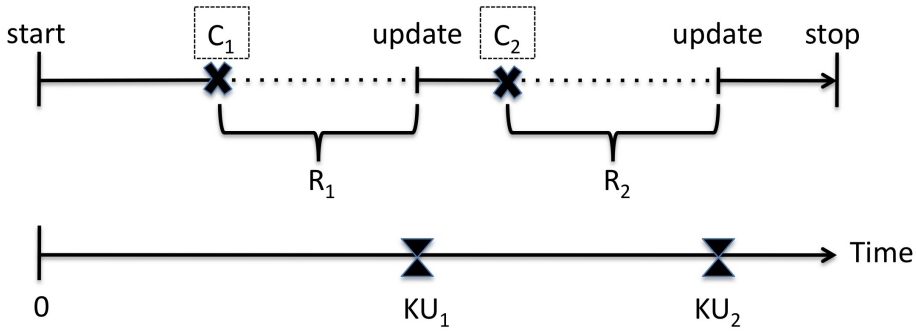


Figure 6.4: Mean Time to Recover

It is important to have an optimum key update strategy and threshold such that MTTR value is as small as possible. However, very small MTTR value requires very high power consumption which might not be acceptable for networks that have devices with limited power. Having this trade-off in mind, now we can start computing the expected MTTR value for our running example: leave-based key update versus time-based key update.

First of all, we need to find the expected total compromised time. This amounts to computing the total time where the key is compromised. Of course, we need a time bound so that we can measure the time from the start of the system (i.e. network) and up to a specified time bound. We can do this measurement by assigning rewards (we don't distinguish between rewards and costs, as in Prism) to the states where the key is compromised. We will call this measured time as *recovery* time, because it is the period where the key waits for recovery (that is a key update). Note that, ironically at the same time it is the period where the communication in the network is insecure. Here is the reward structure for recovery:

```
rewards "Recovery" Comp:  1; endrewards
```

Now that we have computed the total compromised time, the next thing to do is to compute the number of key compromise events in the same period of time where we observe the network. This time we will use a transition reward instead of a state reward. We try to capture all the key compromising transition, technically it requires the key to be fresh (i.e. not compromised) and a key compromising leave transition. Here is the reward for what we call a *new* compromise:

```
rewards "Compromise" [leaveC] !Comp:  1; endrewards
```

The nuance of a *new* compromise will result in different cumulative rewards for different key update thresholds, even though the probability distribution is the same. It seems obvious that, smaller threshold values would result into smaller numbers of (new) key compromises. However, that is a delusion, a false belief that we shed light on Section 6.2.4.

Thus, we have assigned a reward of 1 to each state where the key is compromised, and each transition that causes a fresh key to be compromised, and we are able to store them using the reward structures *Recovery* and *Compromise* above. Now we can proceed to specify the property to compute the rewards that will return us the MTTR value. Below is the CSL formula that we use for this series of experiments:

```
R{"Recovery"}=?  [ C<=30*T ] /
R{"Compromise"}=?  [ C<=30*T ]
```

Actually, the CSL formula above is a combination of two CSL formulas. In the first line, we specify the *cumulative reward* property for recovery up to the time bound of `T` months, and in the second line we do so for key compromise. We can make the division operation at once to get the MTTR immediately, but it is also useful to use the CSL formulas separately to get insight about the recovery times and number of key compromises for a specific setting.

We present the results for two different key update strategies in Fig. 6.5. Once again, the x-axis shows the month `T` of interest and spans two years period. On the y-axis we have the expected mean time to recover value in terms of days. The solid lines represent different intervals M (or `T_time`) for resetting the key in the *time-based* strategy, whereas the dotted lines represent different thresholds N (or `T_leave`) for resetting the key in the *leave-based* strategy.

Although the results for the time-based strategy are no surprise (i.e. it is always half the key expiry period in days), the ones for the leave-based strategy are not easy to guess without the help of this graph. In terms of recovery, there is a fight between M=1-N=2 (time-based wins!), M=2-N=3 (leave-based wins!), and M=3-N=5 (draw!). We see that for N=4 comparison does not make sense, since M=2 is performing too good and M=3 too bad against N=2.

Another interesting point is, the time-based key update for M=3 follows the leave-based key update for N=5 so closely. As can be seen on top of Fig. 6.5,

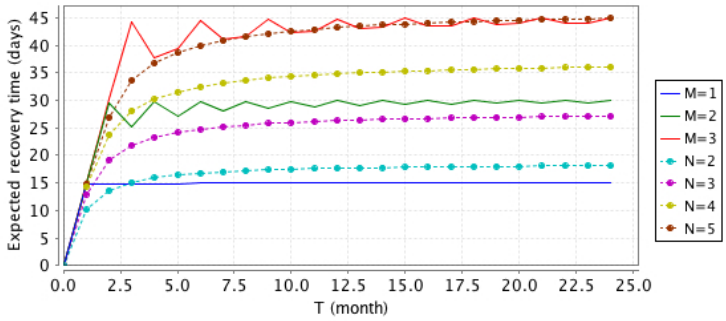


Figure 6.5: Comparison of Mean Time to Recover

time-based key update alternates below and above values of the leave-based key update, for specified thresholds.

6.2.2 Maximum Recovery Time

No doubt, the time needed to recover from this needs to be optimized in such a way that the network is not without protection for a long time period. Therefore, we have explained our approach of MTTR above. However, it is also important that a network does not face a key compromise situation that is longer than a specified time bound. If this is a short time then a potential malicious attacker will not have long time to launch the attack whereas if it is for a long time then the risk will be considerable higher for a successful attack. Thus, we are also interested in the risk that it takes more than a specific amount of time to recover from a compromise of the key. So, for example we may ask the question:

> *What is the probability that the key has not been replaced 3 months after it was compromised?*

Even though the question seems simple, implementing or specifying it in logic is not trivial. Below is the CSL formula that we use to compute the probability that the key recovery takes more than `T` months:

```
P=?  [Comp U>=(30*T) !Comp {Comp}{max}]
```

The formula above translates to the question: *what is the probability of Comp remaining true up until at least* `T` *months elapsed, and eventually being false*

after that. From the two curly brackets atn the end of the formula, {Comp} is known as a *filter* in Prism jargon and used to specify a starting state for the path property (in this case `Comp U>=(30*T) !Comp {Comp}{max}`). However, {Comp} satisfies more than one state therefore we specify a second filter {max} which allows us to compute the maximum probability of the path property from all the states satisfying {Comp}.

Fig. 6.6 shed light on the maximum recovery time of selected key update scenarios. Now the y-axis shows the probability of the key still being compromised after the number of months shown on the x-axis and as above the curves show the result for various choices of parameters.

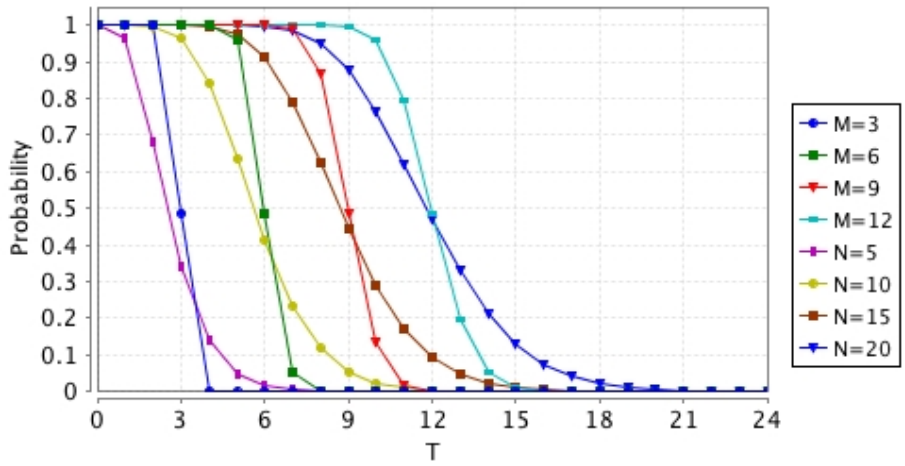


Figure 6.6: The probability that the key recovery takes more than T months.

If the key is replaced every 6 months then the green curve on the figure show that the probability of it still begin compromised after 3 months is 99%. On the other hand, the figure also shows that it is 96.3% if we replace the key whenever 10 devices have left the network. So at this observation point the leave-based approach is the clear winner. However, things change a lot when the observation point changes. Indeed we see that if we change the time frame of interest to be 4 months rather than 3 months then the time-based approach is a clear winner.

The lesson we learned from Fig. 6.6 is, for comparable key update settings (e.g. M=3 N=5, M=6 N=10, etc.), there is a T point where before this time point of interest leave-based key update performs better, and after that point time-based key update performs better. Therefore, a security requirement such as *a key recovery should never take more than* T *months* results in different key update strategy and threshold value selection, depending on the value of T.

6.2.3 Methodology for Optimising Recovery

In the previous sections, we have explained how to analyse recovery from compromised key from two different perspectives. In this section, we present our methodology for determining optimum key update strategy concerning key recovery.

In Table 6.2, we present the methodology as an algorithm of six steps. The idea is to find a solution satisfying networks key recovery requirements. The solution is computed in two sequential steps, making use of the developments we have explained above. Note that the order of the optimisations in STEP-IV and STEP–V can be reversed depending on the priority of either STEP–1.a or STEP–1b.

Similar to the methodology for optimising confidentiality, this methodology is also self-contained. Once again, the requirements maybe refined and/or reset in order to avoid null or too large result sets.

6.2.4 Weakness of Periodical Key Updates

In this section, we would like to point out a weakness in periodical key updates that we relate to the key compromise and recovery.

As we have previously mentioned, expecting less key compromises for smaller key update thresholds is not always the case. Our analysis finds out that in the time-based key update, or so called *periodical* key update, a network can have decreasing number of key compromises even though the key update threshold, or the period is increased.

In Fig. 6.7, we count the number of key compromises using the reward structure and the logic formula that we have previously explained. Our aim is to compare the number of key compromises a) for two different key update strategies b) for different threshold values in a key update strategy. We again use the ZigBee Home Automation application profile that we have mentioned in Section 5.2.2, with 50 devices instead of 20 just to magnify the results. And to be in parallel with our running example, we compare the time-based and leave-based key update strategies that we have introduced in Section 5.2.1. To be more realistic, we will use the phase-type probability distribution implementation of the time-based update, as we have done up to now. The value of the shape parameter `k` is taken as 100. For each strategy, we run the model checkings for twelve different threshold values ranging from 1 to 12; and in each case the observation period

STEP I: Set the requirements
1.a) average time to recover from key compromise should be between a_{min} and a_{max} time units
1.b) the risk that key compromise takes more than T time units should be between b_{min} and b_{max}
1.c) the observation time as OT time units

STEP II: Set the capabilities
2.a) the set of key update methods that can be implemented: $KU = \{KU_1, \ldots, KU_n\}$
2.b) the multiset of threshold sets that includes an implementable threshold set composed of positive integers for each key update method in KU: $TS = \{TS_1, \ldots, TS_n\}$

STEP III: Initialize the solution set as:
$SS = \emptyset$

STEP IV: Compute the solution set that satisfies 1.a

For all i in KU do
For all j in TS do
For all k in TS_j do
1. compute $MTTR = \frac{R\{Recovery\}_{TS_j[k]}[C_{\leq \mathtt{OT}}]}{R\{Compromise\}_{TS_j[k]} =?[C_{\leq \mathtt{OT}}]}$
2. if $MTTR \models$ 1.a then $SS = SS \cup TS_j[k]$

STEP V: Refine the solution set such that it also satisfies 1.b

For all i in SS do
1. compute the probability of recoveries longer than T
2. if $P_{SS[i]}[Comp\ U_{\geq \mathtt{T}}\ !Comp\ \{Comp\}\ \ \{max\}] \not\models$ 1.b then $SS = SS \setminus SS[i]$

STEP VI: Return the solution set SS.

Table 6.2: Methodology for optimising recovery

is 5 years.

Fig. 6.7 teaches us two important lessons: *a*) leave-based key update strategy causes less number of key compromises than time-based key update for threshold values less than 9 *b*) we can expect increase of key compromises when we increase

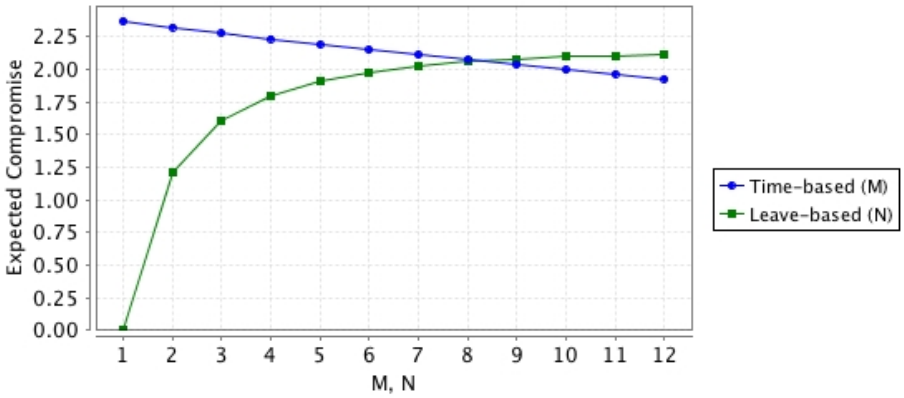


Figure 6.7: Comparison of number of key compromises for time-based and leave-based key update strategies. ZigBee Home Automation profile, 50 devices, 5 years period.

update thresholds in leave-based key update but NOT in time-based key update. The first lesson might be very misleading because as the threshold values get larger, a key compromise event takes much much longer to recover. In other words, for large key update periods the network will remain insecure for long time periods. Therefore, number of key updates won't be an important criterion. Besides, even though they share the same axis the key update thresholds are not comparable. For example, we have previously shown that M=3 and N=5 are comparable in terms of key compromise probability and MTTR. The second lesson is very important, since it is a kind of discovery which would not be easy to find out without formal verification. Intuitively, updating less frequently will be less secure. That still holds, however stochastic model checking shows that number of key compromises in time-based update decreases monotonically as we apply less key updates.

Now let us illustrate the hidden point behind those two lessons. In Fig. 6.8, we give an example including three instances using time-based key update where we keep everything else but the update threshold constant. The time-line on top labelled as `M=1` which means the key is updated monthly. Key updates are denoted by little bars, and since each update is after a month we can see that the observation period is 6 months. Similarly, the time-line below shows key updates every two months, and the last every three months. Since the distributions are the same, we have the key compromise event in the same time point regardless of the key update threshold. The first key compromise is common to all three instances, named as $\mathtt{C}_1$, happens before one month passes after network (or observation of the network) starts. Following the same fashion

as in our previous example, we notice that the dotted part of the time line for this compromise lasts longer as the update threshold increases. In other words, the network key remains compromised for a long time when `M=2` and `M=3`, compared to `M=1`. Then in the second month, we observe another key compromise only in `M=1`. This key compromise is not visible in the other cases since their keys are already compromised. In the end of the second month, we observed two key compromise events in `M=1`, and only one event in `M=2` and `M=3`. Now can we claim that `M>2` is more secure because they have less number of key compromises?

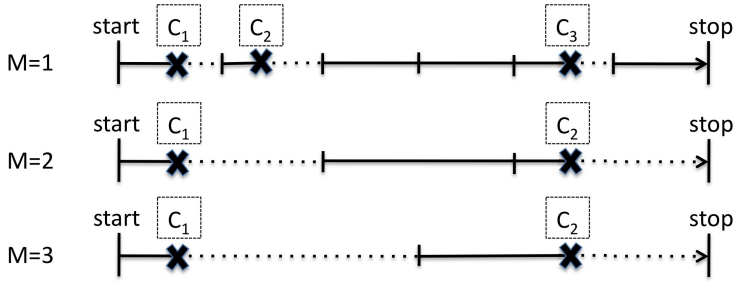


Figure 6.8: Demonstrative number of key compromise for different thresholds in time-based key update

Continuing with the example in Fig. 6.7, we can conclude that in the end of six months

- the total time of possible insecure communication is increasing when the update period is increased
- number of key compromises *may* decrease (but *never* increase) as the update period increases
- mean recovery time, or the time that the network is insecure is increasing as the threshold value is increasing

6.3 Optimising Efficiency of Key Updates

In this section we optimise the efficiency of the key updates. Below we define two measures related to the power consumption and recovery from key compromise. Considering the low-resource nature of wireless sensor networks, we aim to avoid unnecessary key updates as much as possible.Besides, efficiency is a very important criterion to be used with the previous criteria – confidentiality and recovery – when determining the optimum key update strategy in networks.

6.3.1 Power consumption

One of the many important issues related to the key updates is the *power consumption*. Whenever we are replacing the key we are consuming power, and it is certainly interesting to compare the number of key updates performed over a certain period in different settings. Clearly, if we replace the key every 6 months then it is easy to see that we are updating the key twice a year. But how many times are we updating the key a year if we replace it whenever 10 devices have left the network? We can pose this question to our analysis:

> *How many key replacements can we expect to have we performed a year from now?*

In order to compute the number of key updates, a.k.a. replacements, we need to capture all the transitions that cause key update and assign them a reward.

We will call this reward *replacement* and below is the structure needed for the leave-based key update:

```
rewards "Replacements" [leaveC] true:  1; endrewards
```

Notice that, even though it is completely unnecessary, the only change we need to do is to change the action label as `reset` if we need to count the replacements rewards in the time-based model. After the reward structure we can present the CSL (in fact CSRL) formula to compute the expected accumulated number of replacements:

```
R{"Replacements"}=?[C<=30*T]
```

The result of the analysis is shown on Fig. 6.9, for `T` values varying from 0 to 24 months. As above the curves show the result for different thresholds for replacing the key. The y-axis shows the expected number of key updates accumulated over the number of months shown on the x-axis. We can now see that if we replace the key whenever 10 devices have left the network then we can expect to perform 1.5 key replacements the first year. Rather than concluding on which approach is more power-aware than the other, we present which thresholds of which approach is comparable with another. Clearly, Fig. 6.9 shows that the pairs (M=3,N=5), (M=6,N=10), (M=9,N=15), (M=12,N=20) are of the

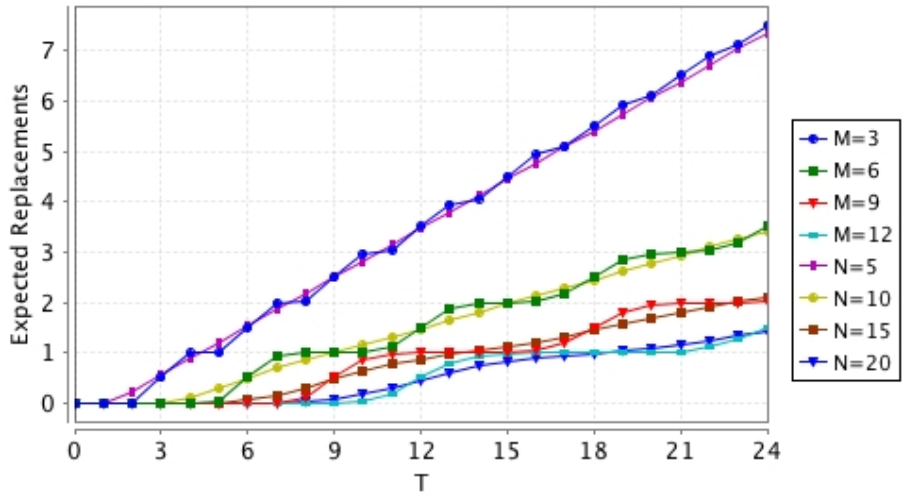


Figure 6.9: Expected number of key updates for leave-based (and time-based) key update strategy. ZigBee Home Automation profile.

comparable settings in terms of key update counts for ZigBee Home Automation application profile.

To demonstrate how this analysis can tighten the bounds of thresholds when looking for the best strategy, we will pose the question:

> *What is the best strategy if we on average can afford 1 key update per year?*

Assuming that we have the input sets as M={3,6,9,12} and N={5,10,15,20} and reusing our analysis in Fig. 6.9, we can eliminate the values that do not satisfy the requirements and choose the lower values of the thresholds and conclude that either M=9 or N=15 is the answer.

However, we have not chosen the best strategy out of this two. In fact, the specific answer depends on the other requirements of the network. For instance, if key confidentiality being strictly below than a percentage is a key requirement then we can use Fig. 6.2 and conclude that leave-based key update for N=15 is better than time-based key update for M=9 because the maximum key compromise is considerably lower (12% for time-based, and 8% for leave-based).

6.3.2 Useful versus Useless Key Updates

In this section, we investigate the extent to which the key updates triggered by the various threshold values are indeed needed, that is, will the key really be compromised when the resets occur. We shall classify the key updates as *useful* and *useless* updates. A *useful update* is a key update that is applied after a key gets compromised, therefore recovering the key. Similarly, a *useless update* is a key update that was not necessary because the key was not compromised. Naturally, we want the percentage of useless updates to be as small as possible because a key update is a costly action for ZigBee devices. We keep the scenarios and the settings of the previous sections, and find out how efficient were our values. To answer this question we shall introduce three transition rewards in our PRISM model as:

```
rewards "All_Resets" [reset or leaveR] true:  1;
                    endrewards
rewards "Useful_Resets" [reset or leaveR] Comp:  1;
                    endrewards
rewards "Useless_Resets" [reset or leaveR] !Comp:  1;
                    endrewards
```

Note that in the reward structures above the name of the reset action depends on the model type, e.g. `reset` for time-based and `leaveR` for leave-based. Next, below are the CSL formulae that we use for specifying steady-state reward properties in this series of experiments:

```
(100 * R{"Useful_Resets"}=?  [ S ]) /
     (R{"All_Resets"}=?  [ S ])
```

```
(100 * R{"Useless_Resets"}=?  [ S ]) /
     (R{"All_Resets"}=?  [ S ])
```

Our approach is instead of using plain rewards, we calculate the percentage of the useful and useless key updates. To manage this we divide the selected update count to all key updates. We get the results for the equilibrium, thus we don't reason about a specific time instant or time bound, but the long run.

As an application, we present the results for the two key update strategies that we have been working on for *Home Automation* scenario In Fig. 6.10. The y

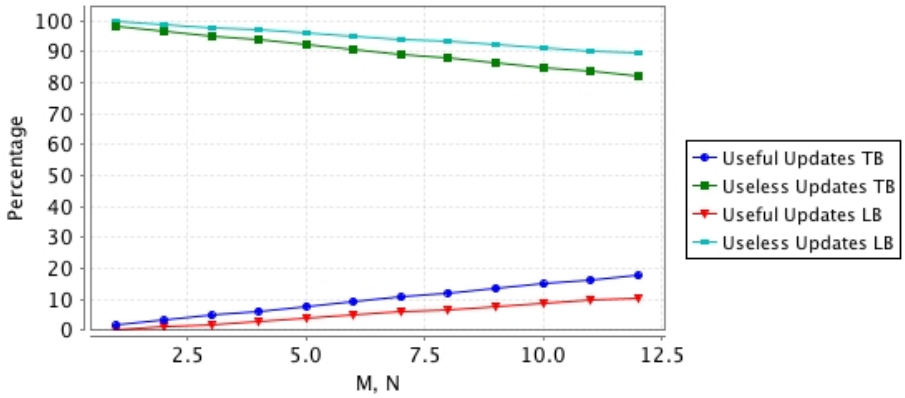


Figure 6.10: Expected number of key updates for leave-based (and time-based) key update strategy. ZigBee Home Automation profile.

axis represents the percentage of resets, and the x axis represents the threshold values (i.e. `M` in months and `N` in devices) for key update. First of all, we must clearly explain that even though thresholds `M` and `N` share the same axis, they are not one-to-one comparable. Namely, `M=5` is very different than `N=5` and no conclusion can be made on which one has more useful. However the results are still very informative and assisting any decision on a set of threshold values. For instance, if we couldn't decide between time-based key update with a threshold of 6 months (TB-6) and leave-based key update with a threshold of 12 leaving devices (LB-12), we can use this analysis and find out that LB-12 has less useless key updates (89.53% vs. 90.77%) and more useful key updates (10.46% vs 9.23%) than TB-6, therefore it is more power-aware. However, if the decision is between TB-6 and LB-10 then the winner is TB-6 with 0.58% difference.

6.3.3 Methodology for Optimising Efficiency of Key Updates

We have explained how to analyse the efficiency of key updates from two different perspectives above. In this section, we present our methodology for determining optimum key update strategy concerning key update efficiency.

In Table 6.3, we present the methodology as an algorithm of six steps. The idea is to find a solution satisfying networks key update efficiency requirements. The solution is computed in two sequential steps, making use of the develop-

STEP I: Set the requirements

1.a) number of tolerated key updates per year should be between a_{min} and a_{max}

1.b) the percentage of efficient key updates should be between b_{min} and b_{max}

STEP II: Set the capabilities

2.a) the set of key update methods that can be implemented: $KU = \{KU_1, \ldots, KU_n\}$

2.b) the multiset of threshold sets that includes an implementable threshold set composed of positive integers for each key update method in KU: $TS = \{TS_1, \ldots, TS_n\}$

STEP III: Initialize the solution set as: $SS = \emptyset$

STEP IV: Compute the solution set that satisfies 1.a

For all i in KU do
  For all j in TS do
    For all k in TS_j do
      1. compute $R\{Replacements\}_{TS_j[k]}[C_{\leq 365}]$
      2. if $R\{Replacements\}_{TS_j[k]}[C_{\leq 365}] \models$ 1.a then $SS = SS \cup TS_j[k]$

STEP V: Refine the solution set such that it also satisfies 1.b

For all i in SS do
  1. compute the percentage of the efficient updates
  2. if $\frac{100 \times R\{Useful\}_{TS_j[k][S]}}{100 \times R\{Useless\}_{TS_j[k][S]}} \not\models$ 1.b then $SS = SS \setminus SS[i]$

STEP VI: Return the solution set SS.

Table 6.3: Methodology for optimising update efficiency

ments we have explained above. Similar to the methodologies for optimising confidentiality and recovery, this methodology is also self-contained.

Part III

Case Studies

Chapter 7

Case Study: Optimal Key Update Strategy

In this chapter, we present a case study on determining optimal key update strategy for custom networks. For three different application domains of WSNs, we demonstrate our method of deriving advice from quantitative analysis that we introduced in the previous chapter.

Besides presenting a case study, we extend our previous models with *key compromise by communication over the network*. In addition, we propose new key update methods that can compete with the previous ones.

7.1 Deriving Advice From Stochastic Model Checking

Obviously, it is not trivial to derive conclusion from the stochastic model checking results on the key update regarding confidentiality, recovery, and efficiency. For instance, the most efficient configuration is not the most secure one. To overcome such dilemma, designers should decide on the priorities of the system and select appropriate security parameters. In this chapter, we give examples on

how to decide on the optimum key update strategy and key update threshold. We start by choosing the application scenario, which depends on the type of the sensor devices and the objectives of the network. To choose the application scenario is fairly easy because ZigBee specification already has specialized application profiles that the designers and developers are supposed to make use of. Then comes the requirements which can be about security, i.e. confidentiality, and performance, i.e. power consumption. Certainly, an extra key update causes an unwanted power consumption and therefore would drain the batteries earlier than expected. After carefully specifying the requirements, we can exploit stochastic model checking on finding answers to our questions. At this point, picking a collection of different key update strategies and a set of threshold values would make everything easier. Model checking results will point us the appropriate threshold values if they exist, and of course different behaviours of different key update strategies. Getting all this information, the rest is evaluating all the solutions for all the requirements and conclude on the solution that satisfies all the requirements.

In the rest of this section, we have two case studies that show how we can get advice from stochastic model checking. In addition, it might be seen as a competition between different key update methods and we see how a method can beat another when different environment conditions and requirements exist.

7.1.1 Case Study I: Personal Area Network

We start by a simple case study, where we can show the power of stochastic model checking in figuring out the best strategy, and also parameter selection for that strategy. In this case study, we assume that we have a *personal* sensor network that is formed to sense health information from a patient and transmit the information to the computer in the nurse station as illustrated in Fig. 7.1. The devices on the patient may vary during the time, since different types of information and monitoring may be needed. Besides, the devices might be replaced because of calibration and battery changing issues. Actually, all these changes correspond to stochastic leave and join events in our models. Such networks can be both used for patients in the hospital or elderly people that needs intensive care. Average life of such a network can be assumed as 6 months to a year. Even though it is a pretty small size sensor network, the information is critical and secure communication is vital.

In three steps we will determine the settings that fit to our requirements:

Step I: We first determine our application scenario. In this case, it is a straightforward and relatively easy task to make the selection of the profile that our

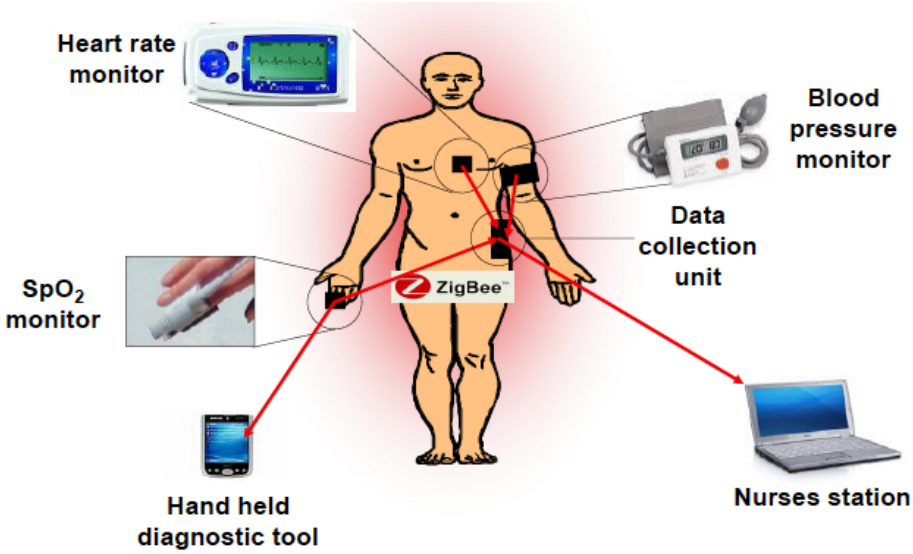


Figure 7.1: Wireless Sensor Network used in Patient Monitoring. *Illustration: courtesy of Phil Jamieson, ZigBee Alliance, 2007.*

application fits into. The application scenario is clearly the *Personal, Home and Hospital Care* scenario (PHHC) as we have mentioned in Section 5.2.2, and derived its features in Appendix B. We will revise the information that is typical to this profile to be customized for this specific patient monitoring case study as below:

- maximum number of devices: *10 devices (though the profile allows up to 500).*
- average join: *1 device per week.*
- average leave: *1 device per week (though the profile says 1 device per month).*
- risk of key compromise: *0.01%.*

Step II: Now, we determine our requirements regarding the security parameters. For the sake of simplicity, we only define one requirement numerically in this case study and it would be about key compromise. The remaining requirements will not include any numerical constraints:

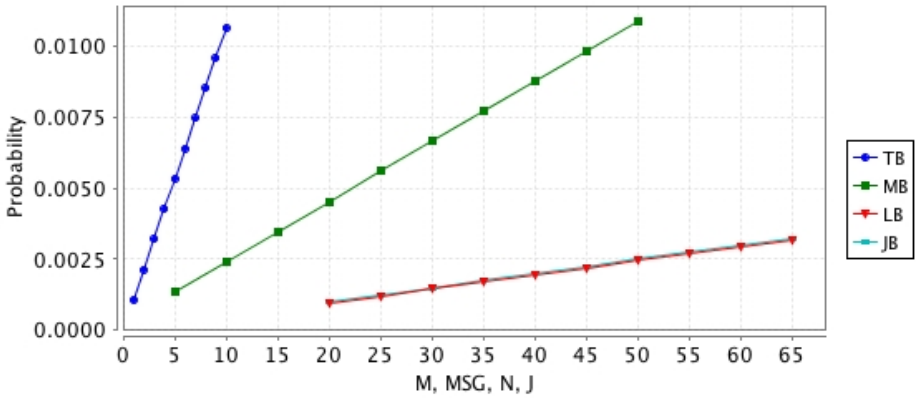


Figure 7.2: Steady-state probabilitiy of key getting compromised in TB, MB, LB, JB strategies having different threshold intervals

- R1 - *the probability that the key is compromised must be lower than 0.25%, at any time.*
- R2 - *when a key gets compromised, the average time for revoking and updating the key should be as small as possible*
- R3 - *since the devices run on battery and too many key updates would drain the battery too quickly, number of key updates should not be more often than what is necessary*

Together with the requirements, we also explain our capabilities. Namely, we need to specify which key update strategies we can implement in the network (specifically in trust center running at the coordinator, and in the devices) and which threshold values we can support. Below are our capabilities for this case study:

- C1 - *Time-based key update (TB) can be implemented using monthly thresholds (M) from 1 to 12.*
- C2 - *Message-based key update (MB) can be implemented using thresholds (MSG) from 5 to 300.*
- C3 - *Leave-based key update (LB) can be implemented using thresholds (N) from 2 to 100.*
- C4 - *Join-based key update (JB) can be implemented using thresholds (J) from 2 to 100.*

Step III: Then, we can determine the parameter values that would satisfy the requirements in the previous steps. To do this, all we need is the results of stochastic model checking. For this simple case study, first we need to check the long run behaviour of the candidate key update mechanisms and then we check the results for specific time instants. We present graphical results for the first part using four different key update strategies and proper (to keep the case study simple we restrict the set of threshold values) threshold value sets in Fig. 7.2. Using this results we can limit the threshold values to {1,2,3} for TB, {5,10} for MB, {25,45} for LB, and again {25,45} for JB, since the rest of the threshold values does not satisfy R1..

As a technical detail, we chose shape parameter `k` as 1000 in the time-based key update, and rate of messages `R_message` as 1/30 meaning that each device sends a message to the network in average once a month. The network in the case study is consisted of devices which are all directly or indirectly connected to the coordinator device (e.g. nurses station in Fig. 7.1) which runs the trust center application that is responsible of key updates and can count the number of messages sent, number of leaving and joining devices, time elapsed, etc.

Next thing to do is to going from long-run to transient probabilities and checking if the requirement is violated at specific time instants. To achieve this, we will continue our work with our restricted set of threshold values and find key compromise probabilities in a reasonable time period. We present the results for this graphically in Fig. 7.3 where the time instant `T` is in months and lines are grouped such that TB (M=1 and M=2) have *circle* points, MB (MSG=5 and MSG=10) have *triangle* points, LB (N=25 and N=45) have *square* points, and JB (J=25 and J=45) have no points on their lines. In fact, Fig. 7.3 tells us that *even though in the long run key compromise probability is below 0.25%, transient probabilities may exceed that limit for settings JB-45 and LB-45.* Although we observe very close probabilities to 0.25% MB-10 (reaches up to 24.85%) and TB-2 (reaches up to 23.9%) are not eliminated in this phase.

Our next focus will be on the mean time to recovery among the remaining key update settings. In Fig. 7.4, we present the expected mean recovery times in days for our set of thresholds. The results reveal that, two of the settings have clearly unacceptable MTTR values, compared to the others. So, we also eliminate the tup two curves, MSG=10 and M=2 from our solution set. After this refinement, we have only one threshold value from each key update strategy.

So far we have touched the key confidentiality, and recovery from a key compromise issues; in other words pure-security related ones. Now we will also consider the power consumption. Up to now, we have eliminated certain settings gradually and now we have a refined set of strategy/threshold values all of which have very close key confidentiality properties. Now, we will evaluate the remaining

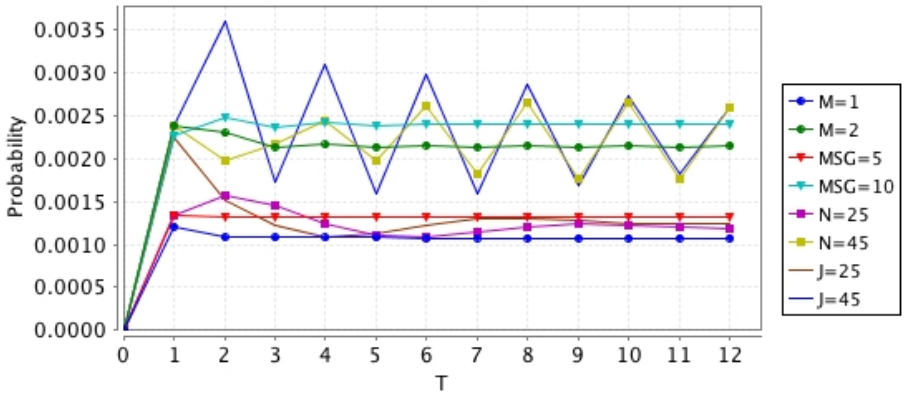


Figure 7.3: Transient probabilitiy of key getting compromised in selected thresholds of different strategies

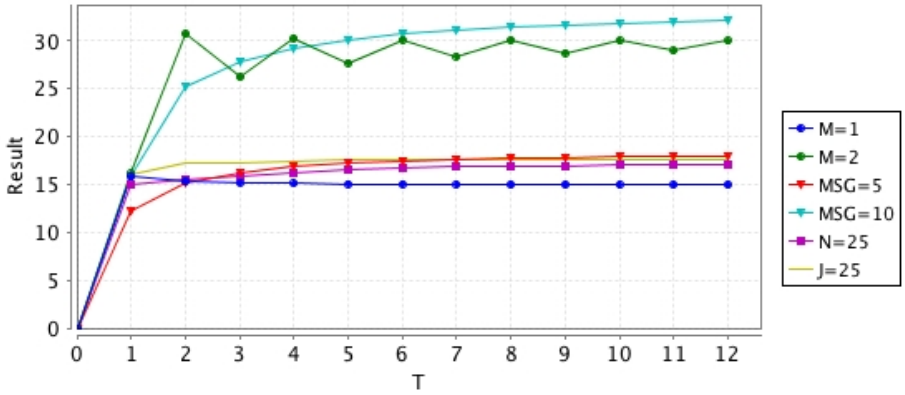


Figure 7.4: Mean time to recover in selected thresholds of different strategies

four settings on how much power they consume when doing this security updates. We will abstract the notion of power consumption in a way that each key update operation cost a certain amount. We will assume that all strategies consume the same amount of energy, and therefore we will consider a setting where we have less number of key updates as the most power-aware setting. We present our related analysis results in Fig. 7.5, where we have expected number of key updates in a cumulative way on y-axis and observation time points as months in the x-axis. We have only given the results for one year of operation, which is already sufficient to derive a conclusion. The top two lines, MSG=5 and M=1, has clearly more key update operations in one year of time and for

sure the difference will be growing as the time passes. Therefore, time-based and message-based key updates are totally eliminated.

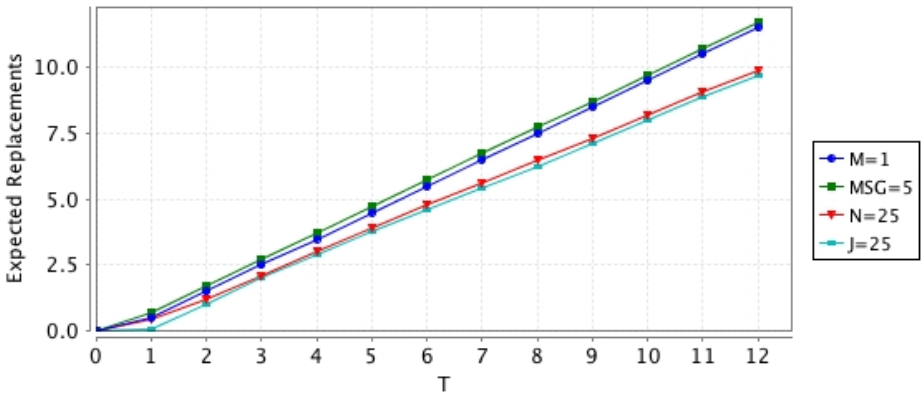


Figure 7.5: Expected number of cumulative key updates at specific time points, in selected thresholds of different key update strategies

As a result, join-based key update with a threshold of 25 devices is the winner key update setting for our scenario. To tell the truth, the difference between leave-based key update with the same threshold value is almost negligible, therefore it is also a winner in this case study.

7.1.2 Case Study II: Marine Asset Tracking system

We can continue with a bit more complicated case study in terms of input values. At the same time, it is more easy and direct to find the solution since the criteria and requirements are set more strictly. This time we will keep the explanations less, and instead stress the data and findings.

In this case study, we have a wireless sensor network that forms a *marine asset tracking system* in a container ship as in Fig. 7.6. Specific containers in a ship that carries sensitive shipments have a sensor device for each, that sends information to the controller computer on the ship. The information is consisting of temperature, intrusion, and location date. We assume that each device sends *one* information package every day. Such a network is used to provide a permanent and freight-specific supervision of each transport package along the supply chain. We assume that a freighter journey can take up to two months, which is the maximum life of the network. Along the way, the container stops at many ports where new containers are shipped and/or old containers are ported.

Figure 7.6: Emma Maersk, largest container ship built until 2006. *Courtesy of A.P. Moller - Maersk Group*

In other words, we have stochastic device leaves and joins. Network security is crucial for the containers that contain sensitive goods and thus having a sensor device, besides a breakdown or power loss in a sensor device is not very tolerable since it is hard to replace it. Depending on the sensitivity of the carried stuff, insider and outsider attacks are expected.

Step I: We choose the *Wireless Sensor Application* (WSA) scenario since asset tracking is clearly specified in this scenario, see Section 5.2.2. The information that we need for model checking is given below:

- maximum number of devices: *200 devices (though the profile allows up to 500).*
- average join: *1 device per week.*
- average leave: *1 device per week (though the profile says 1 device per 6 months).*
- risk of key compromise: *0.1%.*

Step II: Then, we define our requirements.

- R1 - *the tolerated number of key updates in a journey is strictly less than 10.*

- R2 - *the long run probability of a key getting compromised should be lower than 5%.*
- R3 - *the maximum transient probability that the key is compromised must be lower than 10%, at any time.*

After that we present the capabilities in terms of key update mechanisms below:

- C1 - *Time-based key update (TB) can be implemented using **daily** thresholds (M) from 1 to 12.*
- C2 - *Message-based key update (MB) can be implemented using thresholds (MSG) from 100 to 2000 (*incrementals by 100 only*).*
- C3 - *Leave-based key update (LB) can be implemented using thresholds (N) from 2 to 20 (*incrementals by 2 only*).*
- C4 - *Join-based key update (JB) can be implemented using thresholds (J) from 2 to 20 (*incrementals by 2 only*).*

Step III: This time we start from the power consumption requirement which is the first requirement. Also, instead of presenting graphical results we will mention the numerical results as inline text.

To satisfy R1(we make use of the optimising update efficiency approach):

- **Time-based:** In time-based key update, no need for formal verification, the only unaccepted thresholds are 1, 2, 3, 4, 5 and 6 since more than 10 updates would be performed in two months. Therefore we decide to add the three minimum threshold choices to our solution set: TB-7 (8 updates), TB-8 (7 updates), and TB-9 (6 updates).
- **Message-based:** Using simple arithmetic, we can calculate that if each device sends a message a day, and there are 200 devices, than the threshold value should be bigger than 1200 to satisfy R1. However, this result wouldn't hold because 200 is the *maximum* number of devices not the average. In order to verify the threshold bound we use `Replacement` rewards. As a result, we learn that expected number of key updates after 60 days of operation is 8.62 for MB-700, and 10.04 for MB-600. Therefore, our solution set for threshold value is composed of: MB-700, MB-800, and MB-900.

- **Leave-based:** In this case, expected number of replacements are higher than 10 after 60 days of operation when the threshold is less than 88 devices. Therefore, we can choose the minimum acceptable threshold and its two adjacent values: LB-88 (9.90 updates expected according to stochastic model checking results), LB-90 (9.64 updates expected), and LB-92 (9.28 updates expected).
- **Join-based:** In join-based key update, expected number of replacements are higher than 10 after 60 days of operation when the threshold is less than 78 devices. Therefore, we can choose the minimum acceptable threshold and its two adjacent values: JB-78 (9.91 updates expected according to stochastic model checking results), JB-80 (9.64 updates expected), and JB-82 (9.27 updates expected).

To satisfy R2 (we make use of the optimising key confidentiality approach, using steady-state probabilities):

- **Time-based:** Unfortunately none of the TB thresholds satisfy R2 since in the long run TB-7 has 68%, TB-8 has 72%, and TB-9 has 75% key compromise probability. In fact, the first requirement on number of key updates caused TB to produce very high key compromise percentages.
- **Message-based:** In this case, only one of the thresholds satisfy R2 since in the long run MB-700 has 4.87%, MB-800 has 5.55%, and MB-900 has 6.19% key compromise probability. Thus, MB-800 and MB-900 are eliminated.
- **Leave-based:** All LB thresholds satisfy R2 since they have 4.2%, 4.3%, and 4.4% steady-state probability for LB-88, LB-90, and LB-92, respectively.
- **Join-based:** All JB thresholds satisfy R2 since they have 3.8%, 3.9%, and 4% steady-state probability for JB-78, JB-80, and JB-82, respectively.

To satisfy R3 (we make use of the optimising key confidentiality approach again, this time using transient probabilities):

- **Message-based:** The maximum probability at a time instant that the key being compromised can reach to is: 8.5% for MB-700, satisfying R3.
- **Leave-based:** The maximum probability at a time instant that the key being compromised can reach to is: 6.80% for TB-88, 6.86% for TB-90, and 6.89% for TB-92. Therefore, all these three threshold values satisfy R3 as well.

- **Join-based:** The maximum probability at a time instant that the key being compromised can reach to is: 12.6% for JB-78, 13.3% for JB-80, and 13.8% for JB-82. Since all this probabilities exceed 10% limit of R3, join-based key update can not qualify for the solution

In Fig. 7.7, we present the competition between three different key update strategies that qualified until R3. We pick the most promising threshold value for each strategy, and show the key compromise probabilities at different time instants (lines with dots) and also in equilibrium (lines without dots, but having the same color with the corresponding transient probability lines). The most interesting conclusion of Fig. 7.7 is, even though the best performance in the long run is achieved by JB-78 shown by the dotted line with blue color, in the first 10 days it is exceeding the probability limits on the transient probability requirements, therefore it is disqualified at the requirement R3. In the end of the timeline (note that in this figure time unit is a *day*), none of the strategies has stabilized yet; however we observe that the probabilities are converging to the equilibrium. In the comparison of MB-700 and LB-88, which are the remaining settings that qualified all the requirements, the winner is obvious as LB-88 has lower long run probability and the peak value is less than MB-700.

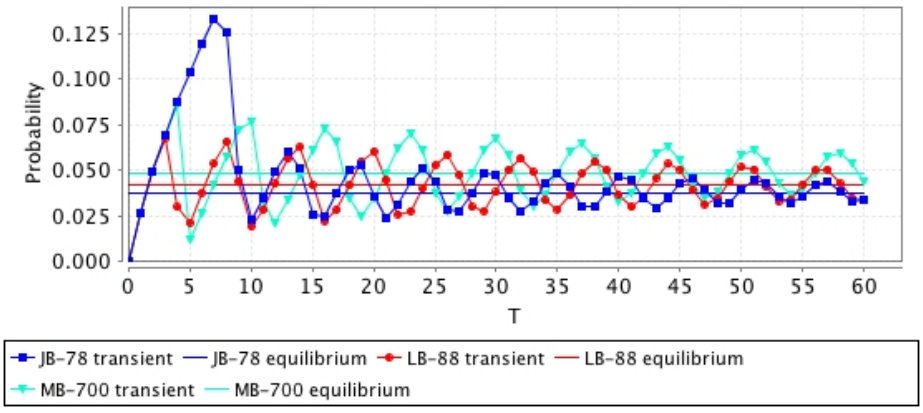


Figure 7.7: Transient and steady-state probabilities of MB-700, LB-88, and JB-78 in Case Study 2

7.2 Improving Key Update Models and Their Quantitative Verification

In this section, first we will round up the key update strategies and the quantitative verification, then we will suggest considerable improvements. The improvements will be on both formal models and key update methods.

7.2.1 Extending the Models with Messaging

Up to now, we used the conventional key update technique that periodically updates the key, we suggested two novel key update strategies that takes the join and leave events in a network into account, and implemented a key update strategy that considers the messages communicated over the network which was proposed by [MAK08].

In the quantitative verification of all these models, we did many abstraction as needed for formal verification. The most important abstraction in terms of security was the key compromise events were only caused by devices leaving or being removed from the network. Of course, this is a very key event and in real life the network will be under serious threat if the same key is being used after a device leave. However, we have excluded the key compromise events caused by communication on the network. Such events could be arising from various reasons such as misuse of key, unintended key transportation, or even attacks based on cryptoanalysis.

Briefly, our first improvement will be on extending all the models to support key compromising (and of course not compromising) messaging. As we have two distinct action labels for leave events, `leave` and `leaveC`, we will have two for messaging events: `message` and `messageC` (and of course if the update is done by message-based strategy then there is also `messageR` action). In Section 5.3 we defined `P_comp` such that a device leaves the network while compromising the key with probability `P_comp`. Both to be compact in the model and not to confuse the developers with too many parameters, we will reuse this probability in compromised messaging events. The idea is, we were assigning probability of `P_comp` to a device leave that tends to be malicious or easily seized by malicious principals; and now we align this to be also the probability of a communicating device that may cause malicious activity. Note that it is trivial to add a new probability parameter and use it in the messaging transitions instead of `P_comp`, and could be done if needed. For example, instead of defining a new probability parameter and significantly increasing the state space, one can use a proportion

of P_comp to be the probability of key compromise by messaging:

```
[leave] Size>0 -> R_leave*(1-P_comp)*Size:  (Size'=Size-1);
[leaveC] Size>0 -> R_leave*P_comp*Size:  (Size'=Size-1);
[message] Size>0 -> R_message*(1-(P_comp/100))*Size:  true;
[messageC] Size>0 -> R_message*(P_comp/100)*Size:  true;
```

In the example snippet above, the probability of key compromise by a leaving device is P_comp, however the probability of key compromise by a message of a device is P_comp/100. Similarly, any product or division of P_comp can be reused with no redundant coding or states. Complete listings of improved models are available in Appendix C.2.

7.2.2 New Key Update Methods

Our next improvement will be on directly key update strategies. We will propose two more brand new update strategies. First, we will make use of join and leave based strategies and merge them into a combined strategy that does not distinguish between join and leave events. We name this strategy as *join-leave-based* (**JLB**) key update. Then we will propose the most powerful strategy ever: *the hybrid key update* (**Hy**). In hybrid strategy we will employ all the key update strategies we considered before, and issue the update whenever any of the update counters reach its threshold. All the counters will be reset as the key is updated. In this way, we will be able to benefit from all key update strategies. Naturally, each key update strategy has a different strength e.g. performing good when too many leaves happen, too many messages communicated, environment has less malicious activity, etc.

Join-Leave-based key update. The *join-leave-based* key update strategy is a smart combination of JB and LB, such that there is one counter for joining and leaving devices and a key update is issued when a predefined number of devices has joined or left the network. In other words, we consider each join and leave event as a suspicious event, and do not distinguish between them. JLB offers a better precision on the key compromise probability by adjustment of the update threshold. Besides, the implementation of JLB is not much different nor much costly than the models that inspired it, both in networking-wise and verification-wise.

Hybrid key update. In *hybrid* key update strategy we will have all this strength, with the cost of more computational power. Therefore we will suggest

the hybrid key update to the networks where the coordinator device has sufficient resources. For example, if the coordinator device is a mobile computer or a powerful handheld then we can implement the hybrid strategy. Besides, we implement the first improvement on messaging to both new strategies that we suggest. we explained above. Basically, we modelled two versions of hybrid strategy: one consisting of LB, JB, MB; and other consisting of LB, JB, MB, TB. Notice that the first one is implementable by pure counters which could benefit from hardware and software optimizations. On the other hand, the other version has four different strategies in it therefore we consider that more powerful and use in our case study below. Note that we did not include JLB in Hy, since it is limiting a real hybrid phenomenon and in Hy we want to observe leave and join events separately. Besides, JLB and Hy are two discrete strategies we propose and would like to compare with others.

Characteristics of the key update strategies. In this chapter we have presented a total of six different key update methods that could be used in wireless (sensor) networks. We started by the classical and conventional key update strategy, TB. We used the new method focusing on the number of messages and suggested by [MAK08], MB. We proposed our own key update methods focusing mainly on the joining and leaving devices, JB, JLB, LB, and Hy. Other than the studies in this chapter, we also worked on the characteristics on these key update methods to shed light on how and when to choose and configure them [YNN11a]. We present the characteristics in graphics, and explain the strength and weakness of each method in two different perspectives – networking and verification – in Appendix D.

Below you will read our third case study where we compare the improved models of key update strategies. Again, complete listings of models that realize new key update strategies are available in Appendix C.2.

7.2.3 Case Study III: Commercial Building Automation

In this case study, we focus on a commercial building automation, specifically hotel security management. In the past, hotels with metal room keys routinely lost more than one keys per room every month, leading to an underground market in keys and an increase in theft. Although dramatic advances in technology such as smart cards converted the hotel door locks into intricate electronic systems, those locks and cards are stand alone such that a card needs to be physically transported to the computer in the reception to make any change on its configuration. Besides, theft of the cards or finding lost cards is not considered in classical systems. Using wireless sensors embedded in the locking card, we can benefit from remote cancellation of cards, remote report of door lock status,

remote report of door ajar alarms, etc.

The technical details of such a system in our perspective includes maximum 50 devices in the network, aiming to keep the network in maximum size as much as possible, replacing stolen or broken cards in average two days, each device having non-stop operation for a year in average, each device sending in average one message a day, and probability of a compromise action by either leaving devices or caused by sent messages is one in ten thousand.

Step I: Such a scenario fits in the *Commercial Building Automation* (CBA) scenario however we need to adjust the input parameters since they are specified clearly above. The information that we need for model checking is given below:

- maximum number of devices: *50 devices.*
- average join: *1 device per two days.*
- average leave: *1 device per a year.*
- average messaging: *1 message a day per device.*
- risk of key compromise: *0.01%.*

Step II: In this case study, instead of defining requirements and checking how the strategies satisfy those, we will compare the strategies directly. As we mentioned before, in this case study key compromise not only happens by leaving devices but also the communication over the network. Therefore, all models are different than the ones in previous case studies. Below are the key update strategies that we will compare:

- TB - *Time-based key update with monthly threshold M.*
- MB - *Message-based key update with threshold MSG - since there will be plenty of messaging, we expect good results.*
- LB - *Leave-based key update with threshold N.*
- JB - *Join-based key update with threshold J.*
- JLB - *Join-Leave-based key update with threshold JL - new proposal, we want to see its performance.*
- Hy/mb - *Hybrid key update excluding MB with threshold H as a triple (N-J-M) - new proposal, we want to see its performance.*

Step III: We will compare the key update strategies in Step II in two main criteria:

- Cr1 - *probability of a key getting compromised in the long run.*
- Cr2 - *number of key updates in the end of a year.*

Key compromise in the long run: We start by computing the steady-state probabilities for a moderate set of thresholds. As can be seen in Fig. 7.8, we used the first five threshold values that we can use in the strategies which is {1,2,3,4,5} for TB, LB, JB, JLB; {500,1000,1500,2000,2500} for MB; and {(1-1-500),(2-2-1000),(3-3-1500),(4-4-2000),(5-5-2500)} for Hy/mb. To keep the graphical results more readable, the x-axis is shared by all the key update strategies such that for MB and Hy/mb it is used like index to real threshold value. For example, the point 2.0 in the x-axis means threshold value of 2 for JLB, 1000 for MB, and (2-2-1000) for Hy/mb.

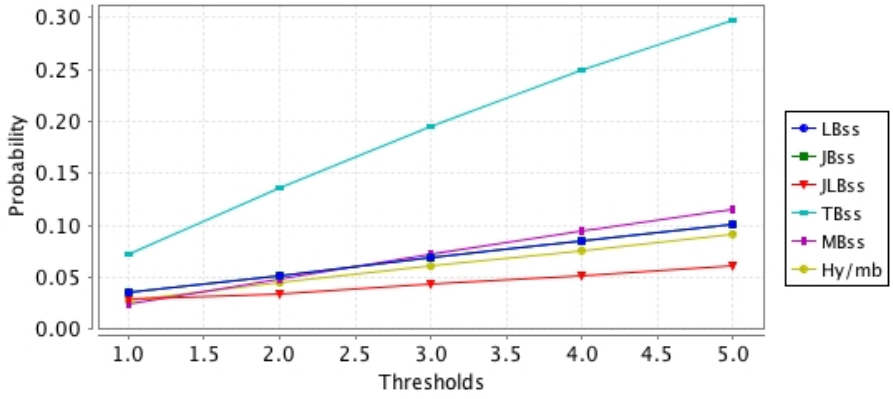


Figure 7.8: Key compromise probability in the long run

Interpretation of Fig. 7.8 is: *a*) TB has very high key compromise probability, *b*) JLB has the best results with only one exception where the threshold values are minimum (and number of key updates are boosted), *c*) the results of JB and LB are so close that green colored line of JB is not visible under LB's blue line; however LB performs slightly better (approximately 0.015% difference in the graph), *d*) JB, LB, MB, and Hy/mb are in a competition where Hy/mb is winning and MB is losing as the threshold values get larger.

Number of key updates: In this criteria, we compute the expected number of key updates after **12 months**. We present the results in Fig. 7.9. On the

y-axis we have the expected number of updates, and as expected the number of key updates should decrease as the thresholds increase.

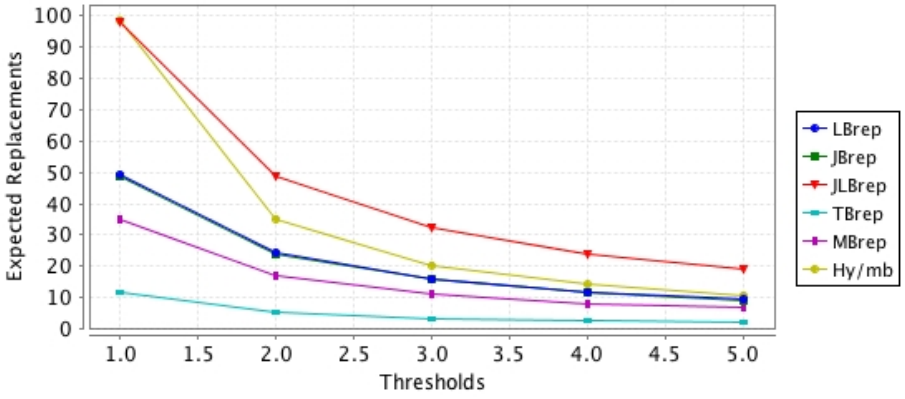


Figure 7.9: Number of key updates in one year of time

Interpretation of Fig. 7.9 is: *a*) JLB has high key update numbers, *b*) TB has the best results, off course we exclude JB in this judgement, *c*) again JB and LB perform very similar, and as expected this time JB performs slightly better, *d*) and again JB, LB, MB, and Hy/mb are in a competition but their results tend to converge as the threshold values get larger.

Minimum key compromise with minimum key updates: Since there is a though trade-off between number of key updates and probability of key compromise, it is not an easy task to find a setting that meets the criteria of minimum key compromise and minimum number of key updates. To keep the case study tractable, we will set two solid requirements here:

- `R1:` The key compromise probability should be below 5% in the long run.
- `R2:` Maximum allowed number of key updates is 35 per year.

Now that we have our specific limits and results above, we can compute the solution sets for each requirement:

S_{R1}={ LB-1,
JB-1,
JLB-1, JLB-2, JLB-3,
MB-1, MB-2,
Hy-1, Hy-2}

S_{R2}={ LB-2, LB-3, LB-4, LB-5,
JB-2, JB-3, JB-4, JB-5,
JLB-3, JLB-4, JLB-5,
MB-2, MB-3, MB-4, MB-5,
Hy-3, Hy-4, Hy-5,
TB-1, TB-2, TB-3, TB-4, TB-5}

Each key update strategy has a solution inside both R1 and R2, except TB which can not satisfy R1 at all. Then it is easy to find the intersection of the solution sets such that:

$S_{R1\&R2}$={JLB-3, MB-2}

As seen above, only two settings satisfied our requirements in the end: Join-Leave-based key update with a threshold of 2, and Message-based key update with threshold of 1000 (remember that MB-2 was actually using threshold of 2x500=1000). Now we can investigate further details, such as transient key compromise probability. In Fig. 7.10, we present the results for key compromise at monthly time instants for JLB-3 and MB-2. This time, we know the steady-state probabilities of these two setting from Fig. 7.8 (0.044 for JLB-3, and 0.048 for MB-2), but we don't know the behaviour of those settings before they get stabilized. Fig. 7.10 supplies good insight and interesting results indeed, such as MB-2 has an alternating pattern where the maximum points increase for almost a year and then start to decrease; at the same time the minimum probabilities decrease for almost two years and then start to increase. In contrast with that, JLB-3 has a very consistent result and seems as the best choice considering that the deviation from the steady-state probability is almost negligible.

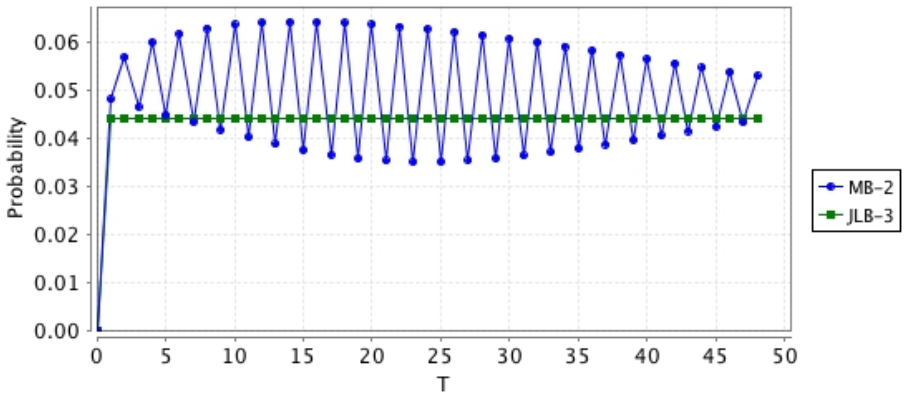


Figure 7.10: Key compromise probability at *monthly* time instants - *comparison of MB-2 and JLB-3*

Although we are happy with the results of JLB-3 where we don't see jumps in the probability, we would like to investigate more thoroughly. Instead of checking monthly time instants, we want to check daily time instant to see more detailed results. In Fig. 7.10, we present the key compromise probabilities at daily time instants. Our concern was to check if there are jumps in the days of a month when using JLB-3. The results verify that even in day level JLB-3 does not

allow peaks, and still very consistent. Besides we can see the details for MB-2, where the probability increases for almost 20 days and exhibits a sharp drop afterwards.

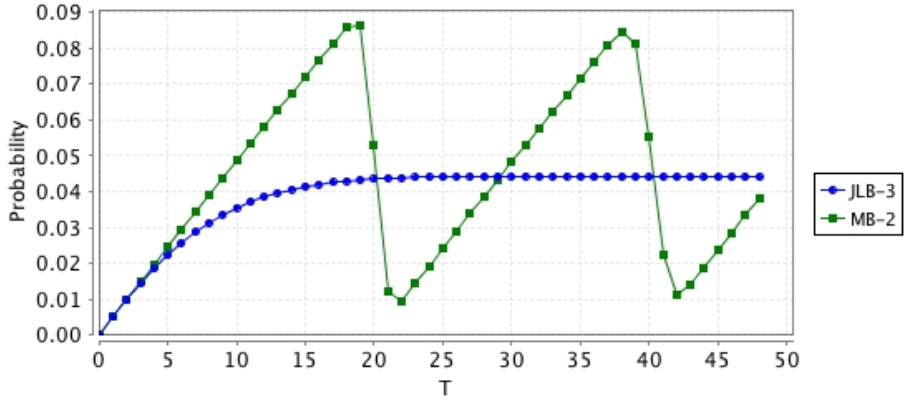


Figure 7.11: Key compromise probability at *daily* time instants - *comparison of MB-2 and JLB-3*

Conclusion: In this case study, we presented a more complex problem that includes key compromise by also communication, and using our approach concluded that Join-Leave-based key update strategy with threshold value 3 is the optimum solution for the key update problem. Another conclusion to be derived from this case study is that both JLB and Hy – our proposed update strategies in this section – perform better than the rest of the update strategies in the sense of key compromise probability. As a side note, we have not seen the power of Hy yet, since we used Hy/mb – a subset of Hy – in case study 3.

Chapter 8

Case Study: Comparison of Key Update Methods

In this chapter, we apply quantitative analysis and get insights on the behaviour of different key update methods under changing network conditions. As the methods are not bound to any type of network but can be implemented for all networks that employ cryptography, the results we present in this chapter are very useful in assessing a key update method.

We start by explaining the purpose of this study in Section 8.1. Then we built a systematic analysis in Section 8.2. We present the quantitative results and interpret them in Section 8.3. We evaluate the considered key update methods based on our analysis results in Section 8.4. Finally, we propose a novel adaptive key update mechanism that would improve the usage of the key update methods in Section 8.5.

8.1 Purpose of the Study

In this section, we explain the purpose of the work in this chapter and describe the questions that we would like to be able to answer at the end of this study.

Our basic aim is to observe how different key update methods behave when the settings of the network they are used is changing in different ways. Assuming that we designed and formed a network (e.g. a ZigBee network, as we have used in the whole thesis), and since we needed a secure network we have also designed and implemented necessary security-related protocols and policies.

In order to keep the analysis tractable we have to make some assumptions. First of all, we assume that we are working on a network where symmetric cryptography is used, and the essential key is the network key which is shared by all the members of the network. We assume that the network is dynamic in the sense that new devices can join the network, and present devices can leave the network. Besides the devices can communicate with each other by sending messages over the network. We assume a star topology where all the devices are connected to a main device that controls the network. Hence, all the messages actually pass through the controller. Then comes our assumptions on the key update methods. We work on six different key update methods which we have used up to now: Join-based (JB), Join-Leave-based (JLB), Leave-based (LB), Message-based (MB), Time-based (TB), and Hybrid (Hy) key updates. We formed the hybrid method to be a proper synthesis of JB, LB, MB, and TB.

Next step is the network dynamics that we are going to feed our analysis. We will base our work on different dynamics such as trustworthiness of the devices, communicativeness of the devices, size of the network, and tendency of the devices to be engaged or disengaged to the network. Considering positive and negative changes in all the dynamics, we will have ten different analysis to answer the questions below:

Q1: *What happens when the members of the network become less reliable?*

Q2: *What happens when the members of the network become more reliable?*

In the two questions above, we query the cases where suddenly devices start to cause more/less key compromises by either leaving the network with a valid keying material, or by sending vulnerable messages. Therefore the devices are more/less reliable in the sense of keeping the keying material secret.

Q3: *What happens when the members of the network communicate more?*

Q4: *What happens when the members of the network communicate less?*

In the two questions above, we question the cases where suddenly devices start to send messages more/less frequently then they used to.

Q5: *What happens when the members of the network tend to leave the network more frequently?*

Q6: *What happens when the members of the network tend to leave the less more frequently?*

In the two questions above, we question the cases where the devices start to leave the network in a higher/lower rate then expected or planned.

Q7: *What happens when new devices tend to join the network more frequently?*

Q8: *What happens when new devices tend to join the less more frequently?*

In the two questions above, we question the cases where new devices start to join the network in a higher/lower rate then expected or planned.

Q9: *What happens when we have to expand the network?*

Q10: *What happens when we have to shrink the network?*

In the two questions above, we question the cases where for some reason we must enlarge the network more than we have planned and vice-versa. Here we mean the maximum size of the network, which will implicitly effect the average size of the network as well.

8.2 Constructing An Analysis

As we have specified the key update methods we are interested in and the questions we would like to be able to answer, the next task is to construct an analysis. Assuming that we will make use of our stochastic models that we developed for each of the key update strategy in Appendix C, we will first define the parameter values that we will use in the analysis. This will be in two steps: first we will define the values for the network-related parameters, then we will define the parameters for the key update methods.

Starting with the network-related part, we first defined three levels for parameters: *High*, *Low*, and *Medium*. We present the values that we assign to the parameters for those three levels in Table 8.1. We chose the values aiming a generic scenario where maximum 50 devices exist in the network but this can be extended to 100 devices or narrowed down to 10 devices in special cases. Each device on average leaves the network in 6 months (because of failure, drained battery, replacement, mobility, etc.), however leaving once a month or once a year is also considered. Similarly each device joins the network with the same rate, however joining once a week and once a year is also considered. Each device sends two messages in a month on average, but in the *Low* case this could be once a month and in the *High* case once a day. Finally, each device can compromise the key with a probability of 0.001. Key compromise may happen

either when the device leaves the network or the device sends a message over the network, for simplicity the probability is the same for both cases. We determine the *Low* and *High* values of this probability as 0.0001 and 0.01, respectively.

When we work on the effect of a parameter, we will isolate it by keeping the other parameters constant (that is we assign the *Medium* values to them), and then analyse the parameter we are interested in by using *High* and *Low* values separately.

Table 8.1: Parameters – *value sets*

Parameter	Description	*Low*	*High*	*Medium*
`Max`	Maximum network size	10	100	50
`R_join`	Rate of join per device	1/365	1/7	1/180
`R_leave`	Rate of leave per device	1/365	1/30	1/180
`R_message`	Rate of messaging per device	1/30	1	1/15
`P_comp`	Compromise probability per device	0.0001	0.01	0.001

Then we defined a set of analyses, where we isolated each parameter by keeping others constant and analyse the effects of the *High* and *Low* values assigned to that parameter. We enumerated our analyses from A1 to A10, such that every pair focuses on a single parameter as seen in Table 8.2.

Table 8.2: Analyses – *parameters*

	`Max`	`R_join`	`R_leave`	`R_message`	`P_comp`
A1	Medium	Medium	Medium	Medium	*High*
A2	Medium	Medium	Medium	Medium	*Low*
A3	Medium	Medium	Medium	*High*	Medium
A4	Medium	Medium	Medium	*Low*	Medium
A5	Medium	Medium	*High*	Medium	Medium
A6	Medium	Medium	*Low*	Medium	Medium
A7	Medium	*High*	Medium	Medium	Medium
A8	Medium	*Low*	Medium	Medium	Medium
A9	*High*	Medium	Medium	Medium	Medium
A10	*Low*	Medium	Medium	Medium	Medium

Now that we have constructed the backbone of the analyses and the values for the network-related parameters, the next job is to determine key update parameters. Actually, the only parameter that we consider in key update is the *update threshold*. To preserve the fairness, we try to find threshold values that produce very close key compromise probabilities in different key update strategies. To find this value for each key update strategy, we compute the probability of the network key being compromised *in the long run*. Using the

models of the key update methods, configured with the parameter values specified as *medium* above (see Table 8.1) we can easily find the thresholds for a fair comparison. The numerical values of the thresholds, and the steady-state probabilities they produce are given in Table 8.3. Note that it is not easy to get exactly the same steady-state probabilities since a minimum increment (or decrement) in the threshold values result in different magnitudes of increments (or decrements) in the probability. For example decreasing LB by 1 will result in 0.018, on the other hand increasing by 1 will result in 0.03. Even though the difference between the results in Table 8.3 may be up to 0.003, we will balance the results by analyzing the difference with the original probabilities.

Another point is, we wrote the threshold for Hy as a set because it is actually a combination of thresholds of $JB = 5$, $LB = 5$, $MB = 75$, and $TB = 1.5$. Thus, hybrid key update model is composed of all other methods in this analysis except JLB. As we have explained in earlier chapters, the reason for this exclusion is to be able to distinguish between leaving and joining devices, and also not to overload hybrid method.

Table 8.3: Thresholds

Method	**Threshold**	$\mathtt{S}_{=?}[Comp]$
JB	3	0.025
JLB	7	0.025
LB	3	0.024
MB	50	0.026
TB	1	0.026
Hy	{5, 5, 75, 1.5}	0.027

The next task is to define the dimensions of the analyses, in the sense that what kind of results we will try to get. We chose three dimensions: *steady–state behaviour*, *transient behaviour*, and *performance* as we present in Table 8.4.

The first dimension, is highly related to the confidentiality of the *network* in the *long run*. We emphasize the *network* instead of *network key* because, as we have previously explained, in the network keying scheme a compromise of a key or a device actually compromises the whole network. This dimension gives very important and very fundamental results, and but it should be assisted by the second dimension, transient behaviour, to allow more thorough analysis. In this dimension, we question time instants instead of the long run. The reason for applying transient analysis is, even though we know the probabilistic results in the long run, we still would like to know about the behaviour before the network reaches a steady–state. At this point, we have to remind the reader that since the time unit in our models is *one day*, and we would like to check for instance *monthly* time instants, than we have to substitute `T` in the logic

Table 8.4: Dimensions of the Analysis

Dimension	Logic formula	Description
Steady–state	$\mathtt{S}_{=?}$ `[ Comp ]`	What is the probability of network being compromised, in the *long run*?
Transient	$\mathtt{P}_{=?}$ $\mathtt{[F}_{[T,T]}$ `Comp]`	What is the probability of network being compromised, at time instant T
Performance	$\mathtt{R\{Updates\}}_{=?}\mathtt{[C}_{<=T}\mathtt{]}$	What is the expected number of key updates until time instant T?

formulae with `30*T`. Then comes the performance dimension, which is highly related to power consumption caused by a key update. In this dimension we accumulate the expected key updates in a reward structure that we named `Updates` and embedded in the formal models depending on the key update method. The details of *Updates* for each key update method is given in Table 8.5.

Table 8.5: Reward Structure *Updates*

Method	Reward Structure
JB	`[joinR] true: 1;`
JLB	`[leaveR] true: 1;` `[joinR] true: 1;`
LB	`[leaveR] true: 1;`
MB	`[messageR] true: 1;`
TB	`[reset] true: 1;`

Hy	`[joinR] true: 1;`
	`[leaveR] true: 1;`
	`[messageR] true: 1;`
	`[reset] true: 1;`

One last thing to do before starting stochastic model checking work is to define the period of observation for the dimensions except steady-state. We defined the period of observation as 16 months, since we discovered in our pre–analysesthat over this period no significant (e.g. visible) change occurs in the probability graphs. Therefore, we will have 16 different model checkings for each key update strategy in each analysis.

8.3 Quantitative Analysis Results

In this section we present the quantitative results of our work. We will categorize analyses in alterations, and for each alteration we will discuss three dimensions as we have introduced earlier.

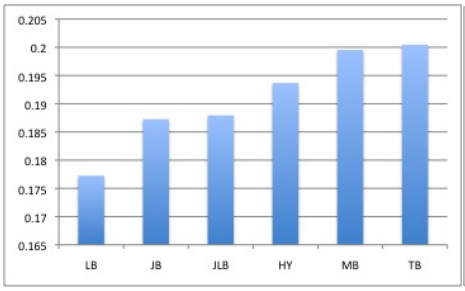
8.3.1 Alteration of Reliability

In this part of analysis, we observe how an alteration in the reliability of the devices impacts the results of key updates. In practice, we observe the effect of key compromise probability of the devices. This part actually consists of analyses $A1$ and $A2$ in Table 8.2, and we are working towards answering the questions **Q1** and **Q2** that we defined in Section 8.1.

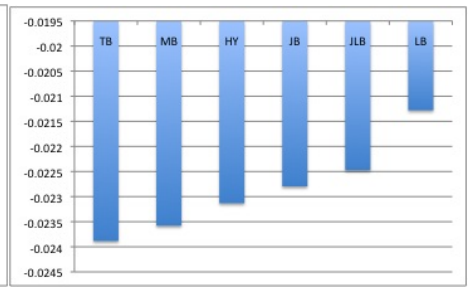
We assume that we have a network that we have formed and configured in some certain configuration. We have set the key update method and key update threshold according to our plans and policies. However, after some point a characteristic of the network changed: *the devices started to behave less/more reliable*. Namely, the probability of causing a key compromise became higher/lower. Therefore, we want to know how our key update strategy (i.e. key update method and key update threshold) will handle this situation. We are interested in the magnitude of the change in steady–state behaviour, transient behaviour, and performance.

Steady–state behaviour. We start our analysis on the alteration of reliability by computing the steady–state probability of the network being compromised. The alteration can be in two ways as we have described before: devices becoming less reliable so key compromise probability per device takes the *Low* value, or devices becoming more reliable so key compromise probability per device takes the *High* value. Since our initial (namely taking *Medium* values) steady–state probabilities were not uniform for all the key update methods, we will present the *difference* in the steady–state behaviour, instead of giving exact probabilities. Though, computing the exact probabilities is as easy as adding these values to the medium values in Table 8.3. We present the results in Fig. 8.1, where the bars indicate the difference in the steady–state behaviour of each strategy, and the y-axis gives the magnitude as probability. Obviously, a positive difference means that the alteration caused the network to be less secure and a negative difference means that the network became more secure.

Interpretation of the steady–state behaviour results: Fig. 8.1 shows that, if the reliability gets lower per device, then LB would adapt very nicely and allow



A1: *High* key compromise probability



A2: *Low* key compromise probability

Figure 8.1: The steady–state behaviour – *Alteration of Reliability*

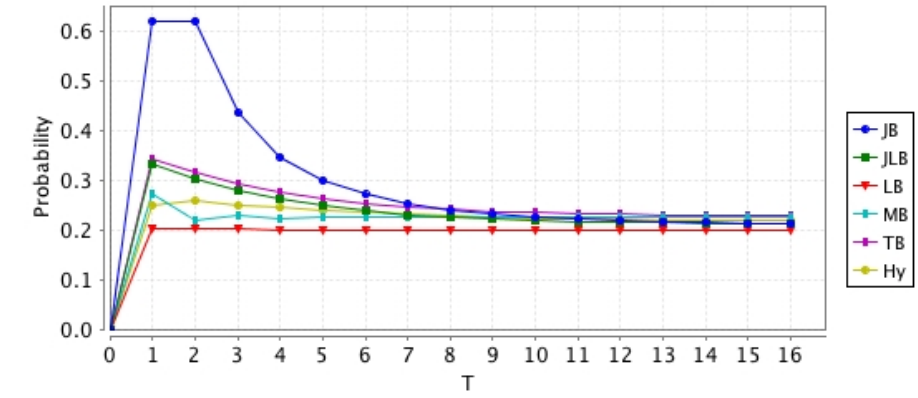
minimum raise in the overall key compromise probability. However, MB and TB will be the worst performers and cause a rather high key compromise. But if the direction of the alteration is the other way round, namely devices get more reliable, then the scene is reversed making TB and MB the best and LB the worst in adapting the alteration in terms of probability of the network getting compromised in the long run.

Transient behaviour. As we know the steady–state behaviour now we can investigate the transient behaviour of the system and thus get more insight. In Fig. 8.2, we present the results for all strategies including monthly time instants for sixteen months. We observe a stabilisation phase of up to 8 months before converging to the steady–state.

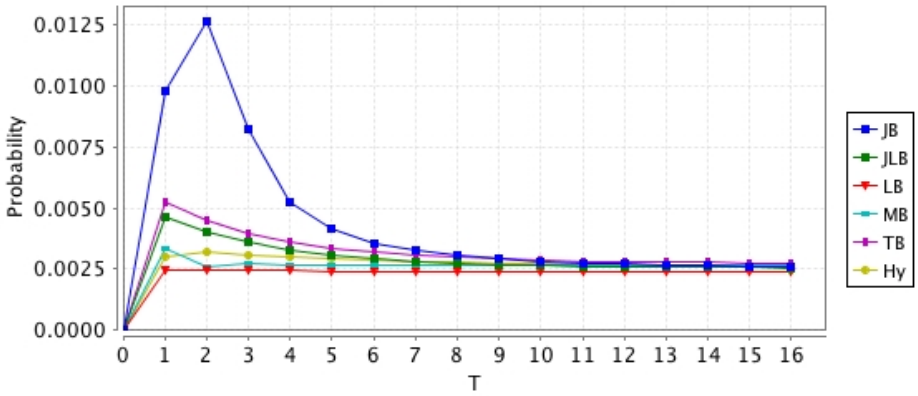
Interpretation of the transient behaviour results: In the steady–state behaviour we observed that different key update strategies adapt differently to the alteration, however the difference is not very big. In other words, there was no jump between the results and the results were limited in the short intervals {0.20, 0.23} for A1, and {0.0024, 0.0027} for A2. In Fig. 8.2, we present that indeed after a stabilization phase the strategies converge to these intervals. However, this stabilization phase is very risky for JB strategy where the overall risk becomes almost three times higher for A1 and almost five times higher for A2. Even though not as risky as JB, the methods TB and JB are also very insecure in the stabilization phase.

Performance. In this dimension, we compute the change in the expected number of key updates in each strategy. For this specific analysis, we are not presenting any graphical results since we have observed the same number of key update as the one before alteration.

Interpretation of the performance results: As the number of key updates in



A1: *High* key compromise probability



A2: *Low* key compromise probability

Figure 8.2: The transient behaviour – *Alteration of Reliability*

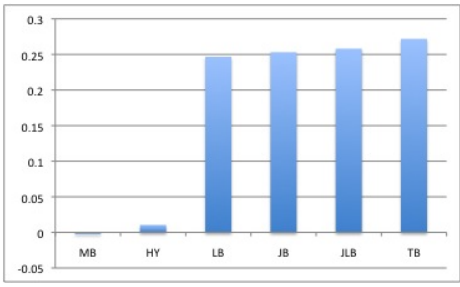
the methods we are interested in are triggered by either number of leaves, joins, messages, time, or all of them, it is no surprise that we will have the same number of key updates after alteration. This means that the power consumption and as a result performance won't be affected by the alteration. In fact, we assume that no key update method can be aware of a key compromise before an attack happens.

8.3.2 Alteration of Communication

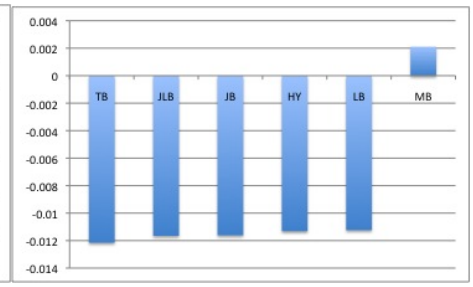
In this part of analysis, we observe how an alteration in the communication of the devices impacts the results of key updates. In practice, we observe the effect of message rate of the devices. This part actually consists of analyses $A3$ and $A4$ in Table 8.2, and we are working towards answering the questions **Q3** and **Q4** that we defined in Section 8.1.

As we have done in the previous section, we analyse the network after a certain alteration happened: *the devices started to send (and receive) less/more number of messages*. Below we present tour analysis in three dimensions in turn.

Steady–state behaviour. As we have done previously, we compute the steady–state probability of the network being compromised. This time the alteration can be in two ways as: devices communicating more so message rate per device takes the *High* value, or devices communicating less so message rate per device takes the *Low* value. Once again, we will present the *difference* in the steady–state behaviour, instead of giving exact probabilities. We present the results in Fig. 8.12, reminding a positive difference means that the alteration caused the network to be less secure and a negative difference means that the network became more secure.



A3: *High* message rate

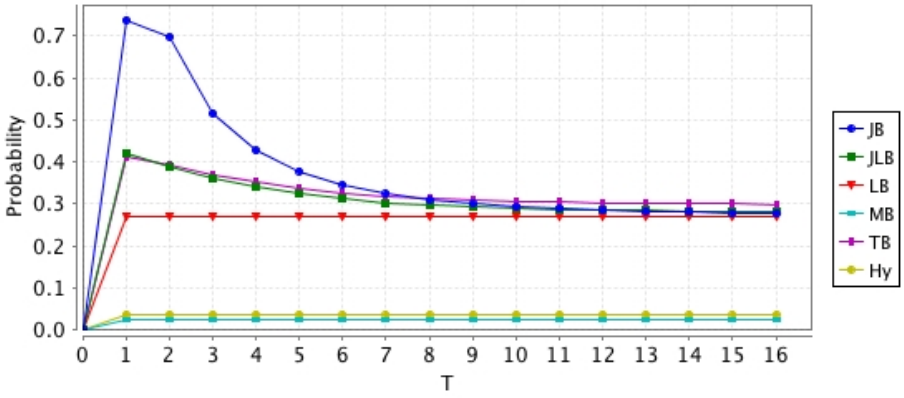


A4: *Low* message rate

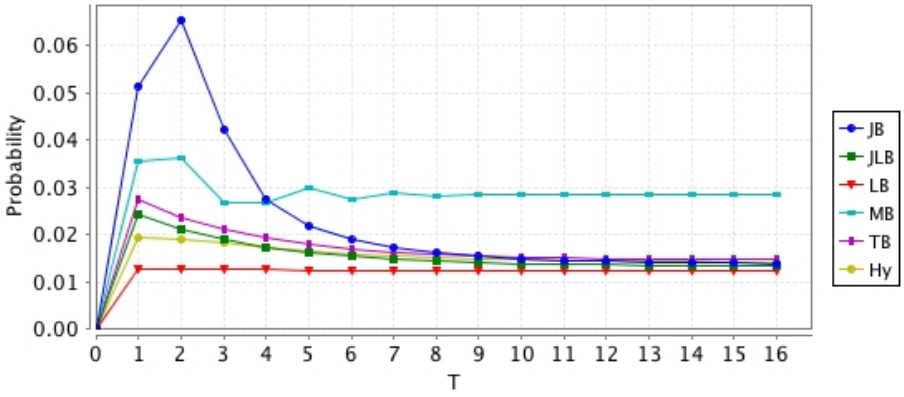
Figure 8.3: The steady–state behaviour – *Alteration of Communication*

Interpretation of the steady–state behaviour results: Fig. 8.12 shows that, if the devices start to communicate more frequently, then MB would adapt very nicely and will provide an even more secure network in terms of key compromise probability. Hy also adapts very well, though slightly increasing the risk. However, all the rest will be the worst performers and causing a very high probability of key compromise. But if the direction of the alteration is the other way round, namely devices start to communicate less than it was before, then the results are very different. This time MB adapts badly and all the rest of the strategies perform very well, and make the network more secure in the long run.

Transient behaviour. We present the transient behaviour of the key update strategies in Fig. 8.4. Again we observe a remarkable stabilisation phase before the probability of compromise converges to the steady–state.



A3: *High* message rate



A4: *Low* message rate

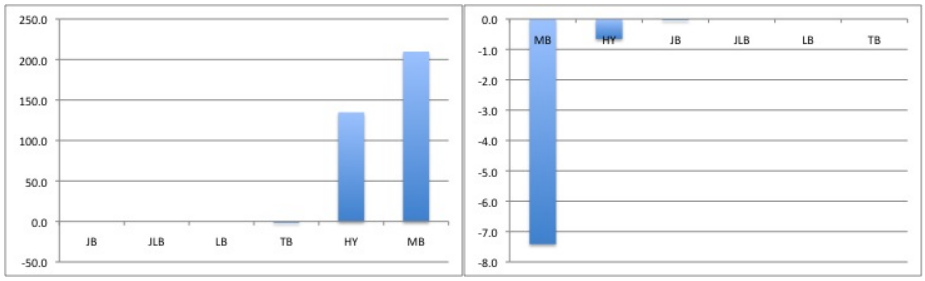
Figure 8.4: The transient behaviour – *Alteration of Communication*

Interpretation of the transient behaviour results: When the message rate is high, Hy and MB have the best results for all the period and all the rest of the key update methods converge to a very close result which is much higher compared to Hy and MB. Besides, JB, JLB, and TB has high key compromise probability before the stabilisation. LB is very consistent and produces no jumps at all.

When the message rate is low, all the methods except MB converge to a close result. The worst strategy in this case is MB which is no surprise when number

of messages is low. Again, the stabilization phase is risky in many update strategies but mostly in JB.

Performance. In this dimension, we compute the change in the expected number of key updates in each strategy. We present the graphical results, as a difference from the performance before alteration in Fig. 8.5. The observation period is *one year*.



A3: *High* message rate A4: *Low* message rate

Figure 8.5: The performance – *Alteration of Communication*

Interpretation of the performance results: In the case that the communication is altered in a way that devices send more messages, Hy and MB start to produce much higher number of key updates. As we present in Fig. 8.12, Hy produces 135 more key updates and MB produces 210 more key updates. Not presented in the figure but can easily be computed that the number of key updates before alteration was only 14 in total in a year for both MB and Hy, thus the performance is very negatively affected. Obviously, in this case the battery of any sensor device will be drained far too quickly. In the rest of the update methods, we don't observe significant changes in the number of key updates.

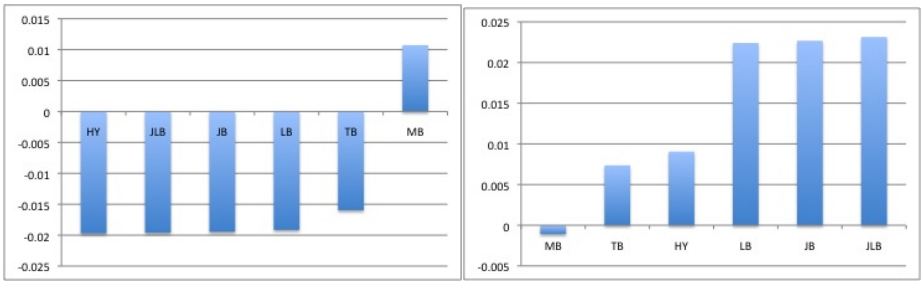
In the case that the communication is altered in a way that devices send less messages, MB starts to produce much less number of key updates. In the perspective of power consumption, it is very good and will prolong the battery life. Hy also performs well, though not as remarkable as MB. All the remaining methods are neutral in terms of performance.

8.3.3 Alteration of Leave Rate

In this part of analysis, we observe how an alteration in the leave rate of the devices impacts the results of key updates. This part actually consists of analyses $A5$ and $A6$ in Table 8.2, and we are working towards answering the questions **Q5** and **Q6** that we defined in Section 8.1.

As we have done in the previous section, we analyse the network after a certain alteration happened: *the devices started to leave the network less/more frequently.* Below we present tour analysis in three dimensions in turn.

Steady–state behaviour. In parallel with the previous alteration analyses, we present the results for the steady–state behaviour in Fig. 8.6. In both cases of alteration, we observe both positive and negative changes in the results.



A5: *High* leave rate A6: *Low* leave rate

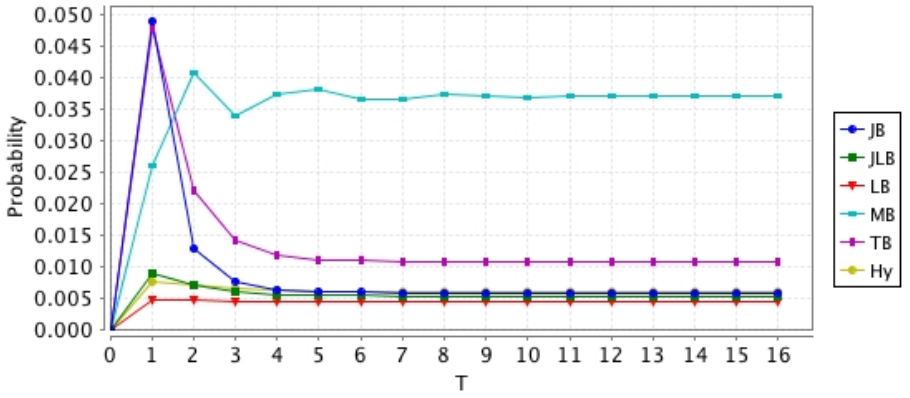
Figure 8.6: The steady–state behaviour – *Alteration of Leave Rate*

Interpretation of the steady–state behaviour results: Fig. 8.6 shows that, if the devices leave the network more frequently than they were doing before, all the methods except MB would provide a more secure network in terms of key compromise probability. But if the direction of the alteration is the other way round, namely devices start to leave the network less than it was before, then the results are very different. This time all the methods that include key updates triggered by leaving devices (i.e. JB, JLB, and LB) cause much higher risk. TB and Hy also causes more risk, but much less compared to JB, JLB, and LB. MB surprisingly performs even better when the leave rate of the devices decrease.

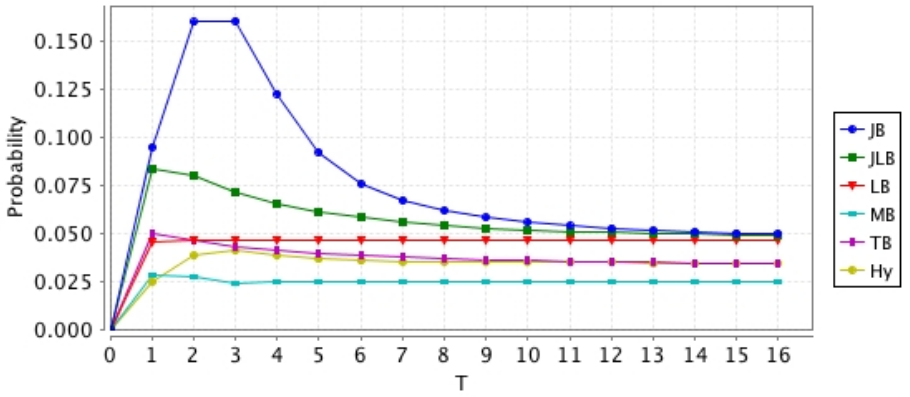
Transient behaviour. We present the transient behaviour of the key update strategies in Fig. 8.7. Again we observe a remarkable stabilisation phase before the probability of compromise converges to the steady–state, however the duration of this phase is very much dependent on the type of the alteration.

Interpretation of the performance results: When the leave rate is high, all the methods except MB and TB converge to an acceptable result and since the difference in the results are minimum they can all be considered as winners. TB is still acceptable but MB is definitely not a good choice in this case.

When the leave rate is low, MB is the winner followed by TB and Hy. Then rest of the methods converge to a very close number.



A5: *High* leave rate



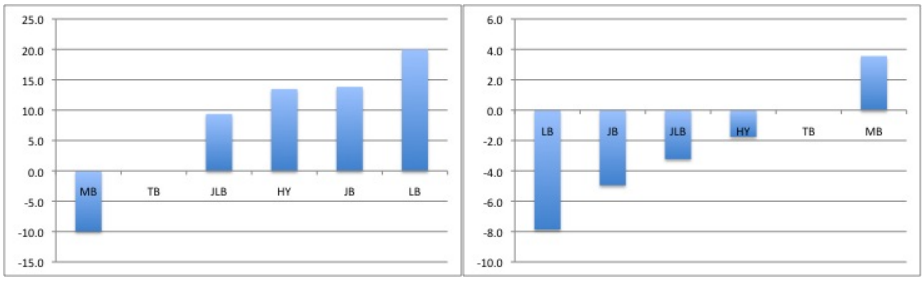
A6: *Low* leave rate

Figure 8.7: The transient behaviour – *Alteration of Leave Rate*

In both cases, stabilisation phase reveals out that even though in the long run the probability of the key being compromised will be low, JB will have high probabilities before the stabilisation. The same goes for TB for high leave rate, and JLB for low leave rate.

Performance. In this dimension, we compute the change in the expected number of key updates in each strategy. We present the graphical results, as a difference from the performance before alteration in Fig. 8.8. The observation period is *one year*.

Interpretation of the performance results: In the case that the network is altered in a way that devices leave more frequently, all the methods except MB



A5: *High* leave rate A6: *Low* leave rate

Figure 8.8: The performance – *Alteration of Leave Rate*

and TB start to produce much higher number of key updates. TB does not have change in the number of key updates, whereas MB produces fewer updates.

The results are almost opposite in the case when devices leave less frequently. LB is the best adapting method, and has fewer key updates, followed by JB, JLB, and Hy. MB is producing more updates than before, in contrast with all other methods.

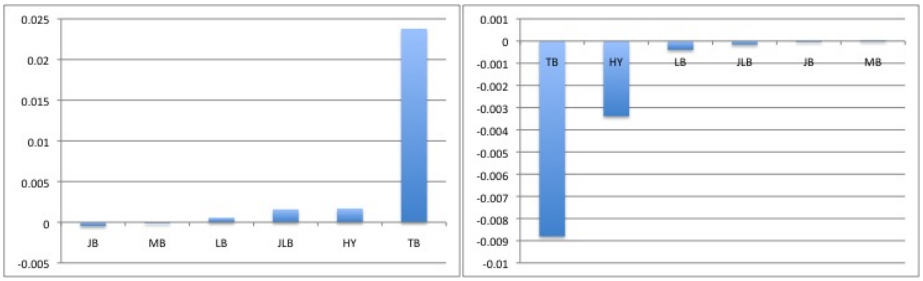
8.3.4 Alteration of Join Rate

In this part of analysis, we observe how an alteration in the join rate of the devices impacts the results of key updates. This part actually consists of analyses $A7$ and $A8$ in Table 8.2, and we are working towards answering the questions **Q7** and **Q8** that we defined in Section 8.1.

As we have done in the previous section, we analyse the network after a certain alteration happened: *the devices started to join the network less/more frequently.* Below we present tour analysis in three dimensions in turn.

Steady–state behaviour. As we have done previously, we compute the steady–state probability of the network being compromised. We present the difference after alteration in Fig. 8.9.

Interpretation of the steady–state behaviour results: When the network is altered such that the devices started to join with a higher rate, JB is is the best performer as expected. Still, the rest of the strategies perform well i.e. not causing significant changes, except TB. Behaving very different than others, TB cause high risk for the network.



A7: *High* join rate A8: *Low* join rate

Figure 8.9: The steady–state behaviour – *Alteration of Join Rate*

In the case of low join rate alteration, the picture changes dramatically. This time TB is the winner, followed by Hy. These two strategies provide higher security than before. Moreover, all the remaining strategies are also adapting nicely, none of them cause increase in the risk.

Transient behaviour. In Fig. 8.10, we present the transient behaviour results when all the parameters are constant except the join rate for a single device. We investigate the effect of high `R_join` value in the upper figure and low `R_join` value in the lower one.

Interpretation of the transient behaviour results: When the join rate is high, all the strategies except TB give very similar results. The method to be avoided is TB this time, which is significantly risky at all times.

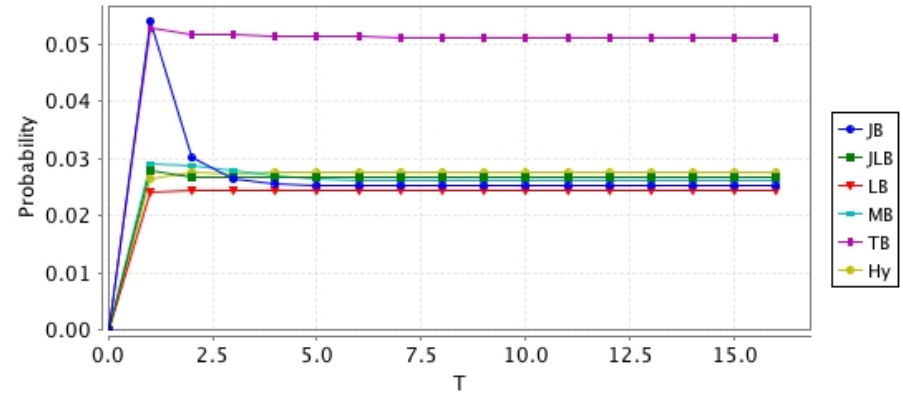
When the join rate is low, all methods produce similar results after the phase of stabilisation. However, in this case stabilization takes a long time compared to the previous case.

In both alterations, JB causes very high risk until it reaches steady–state.

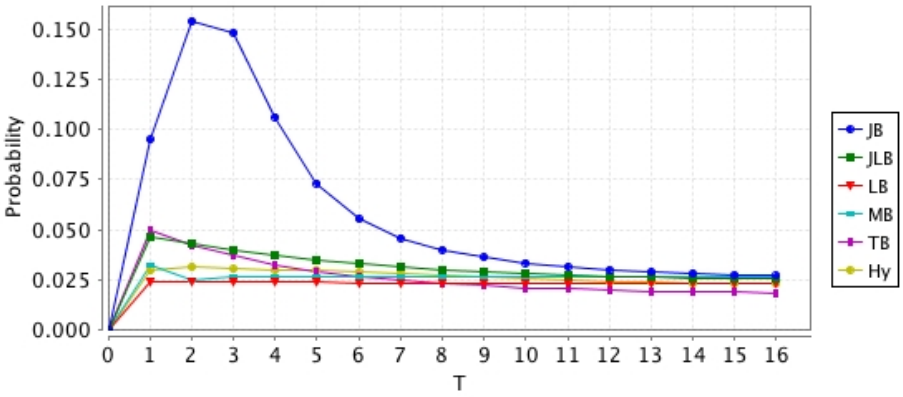
Performance. In this dimension, we compute the change in the expected number of key updates in each strategy. We present the graphical results in Fig. 8.11

Interpretation of the performance results: When the alteration is towards high join rate, than TB does not bring any additional cost which makes it the best performer. Rest of the strategies require many more key updates than before. Especially the strategies based on join events are the most costly one.

When the alteration is towards low join rate, than the order is absolutely re-



A7: *High* join rate



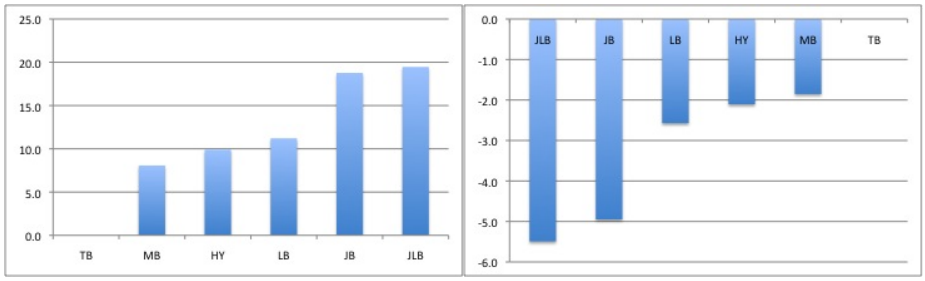
A8: *Low* join rate

Figure 8.10: The transient behaviour – *Alteration of Join Rate*

versed. TB is the worst, even though it does not require any additional cost than before. JLB and JB are the winners since they issue less key updates than before.

8.3.5 Alteration of Network Size

In this part of analysis, we observe how an alteration in the network size impacts the results of key updates. In practice, we observe the effect of the maximum network size which is aimed to be preserved in many cases. This part actually consists of analyses $A9$ and $A10$ in Table 8.2, and we are working towards



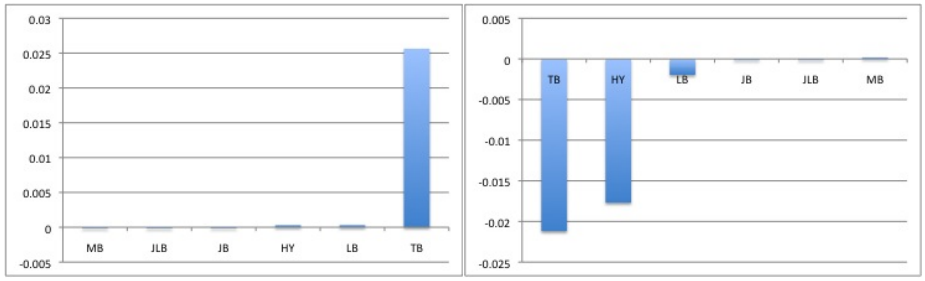
A7: *High* join rate A8: *Low* join rate

Figure 8.11: The performance – *Alteration of Join Rate*

answering the questions **Q9** and **Q10** that we defined in Section 8.1.

As we have done in the previous section, we analyse the network after a certain alteration happened: *the size limits of the network is raised/lowered*. Below we present tour analysis in three dimensions in turn.

Steady–state behaviour. We computed the probability of the network being compromised in the long run after the network is altered. We present the results on the steady–state behaviour in Fig. 8.13.



A9: *High* maximum network size A10: *Low* maximum network size

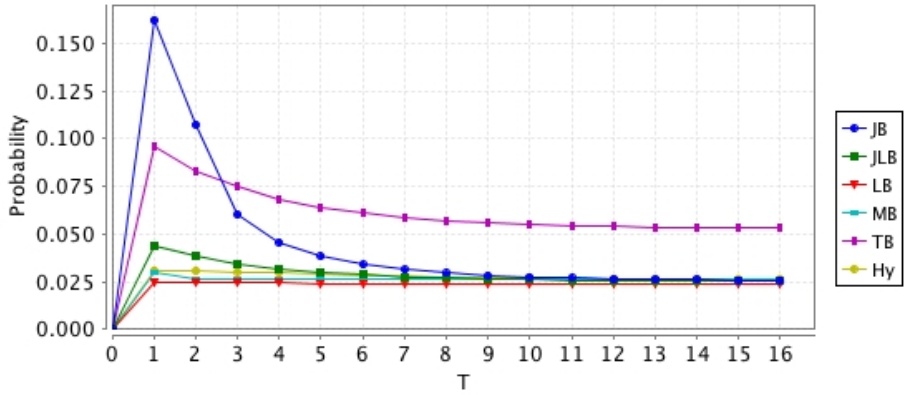
Figure 8.12: The steady–state behaviour – *Alteration of Network Size*

Interpretation of the steady–state behaviour results: We observe that all the key update strategies except TB can very effectively deal with the raise in the maximum network size. The increase in the steady–state probability is negligible for in MB, JLB, JB and very small in Hy and LB (0.0004 for both). However, TB cannot protect the network as effective as it was doing before.

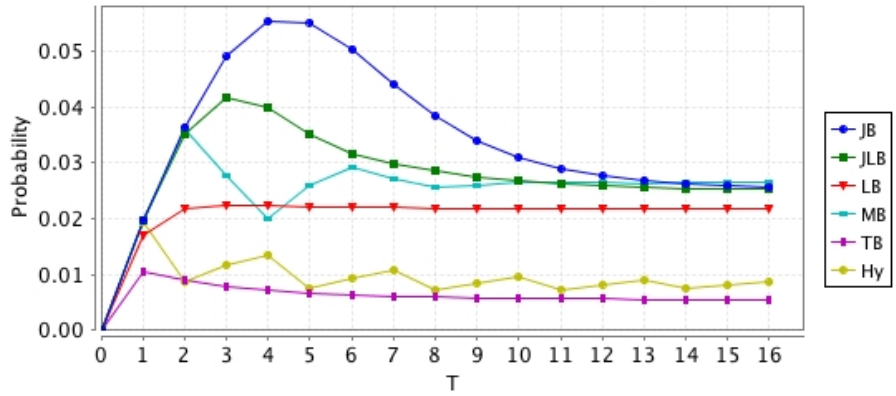
When the alteration is as lowering the maximum network size, TB and Hy provide a better protection. The remaining strategies do not have significant

change in the steady–state behaviour, performing as they were before.

Transient behaviour. In Fig. 8.13, we present the results in transient behaviour such that the effect of high `Max` value is shown in the upper figure and low `Max` value in the lower one.



A9: *High* maximum network size



A10: *Low* maximum network size

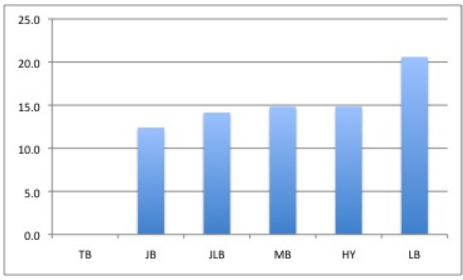
Figure 8.13: The transient behaviour – *Alteration of Network Size*

Interpretation of the transient behaviour results: When the maximum size is high, all the methods except TB converge to a very close probability. TB performs poorly compared to the others, and JB has a high peak value in the first month. MB, Hy, and especially LB perform very well even in the stabilisation phase.

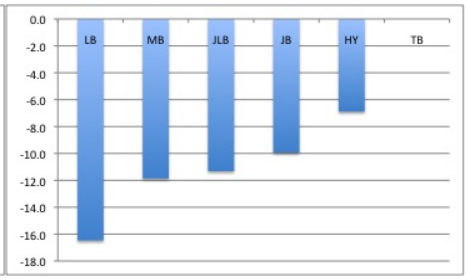
When the maximum size is low, there is a big difference in the behaviours.

Hy and TB are the obvious winners, then comes LB but not very close to the winners. The third group, is JB, JLB, and MB produces very close results however they all have problems in stabilisation.

Performance. In this dimension, we compute the change in the expected number of key updates in each strategy. We present the graphical results in Fig. 8.14.



A9: *High* maximum network size



A10: *Low* maximum network size

Figure 8.14: The performance – *Alteration of Network Size*

Interpretation of the performance results: When the network is altered to be larger in size, all the methods except TB are more costly than they were before. Among them, LB is the most expensive one.

When the network is altered to be smaller in size, the results turn upside down i.e. all the methods except TB are less costly than they were before. Among them, LB provides the biggest saving in terms of number of key updates and thus power consumption.

As a result an alteration in the maximum size of a network affects the performance of all the key update methods that are not *periodical*.

8.4 Evaluation of the Key Update Methods

In this section we evaluate all the key update methods based on our analyses. We tabulate the results of our analyses such that for each dimension a checkmark in the table indicates that the method can be safely used, and a bullet means that though the results are not as good as the ones having checkmark the method can still safely be used. In the end, we merge all the results and present the methods that satisfy all the dimensions.

Dimension 1: *Steady–state behaviour.*

We start with the first dimension: *probability of network being compromised in the long run.* We have ten rows indicating different alterations in the network dynamics, and six columns indicating different key update methods. We use two symbols to denote the security level after alterations:

- **Checkmark:** After alteration the network has even better protection than before, in the long run. In practice, this corresponds to a negative difference between the new steady–state probability and the old one.
- **Bullet:** After alteration the risk that the network will be higher, but the difference will be negligible. In our analyses, we assumed the differences less than 0.0001 as negligible.

We present the results of the evaluation in Table 8.6.

Table 8.6: Analyses – *compromise of network in the long run*

if the devices become	**JB**	**JLB**	**LB**	**MB**	**TB**	**Hy**
less reliable						
more reliable	✓	✓	✓	✓	✓	✓
communicate more				✓		
communicate less	✓	✓	✓		✓	✓
leave more	✓	✓	✓		✓	✓
leave less				✓		
join more	✓			✓		
join less	•	✓	✓	•	✓	✓
expand	•	•		✓		
shrink	✓	✓	✓		✓	✓

Dimension 2: *Transient behaviour.*

We continue with the second dimension: *transient probability of network being compromised* . In this dimension, we are investigating the variation of the risk in the network. We are interested in the period starting from the alteration and lasting until again reaching a steady–state, which we call the *stabilisation* period. The symbols we use have completely different meanings:

- **Checkmark:** The maximum probability of network compromise is not significantly different than the steady–state probability. In other words, we don't observe peaks in the transient behaviour. In our analysis, we

assumed the results where the ratio of maximum probability and steady–state probability was below 1.1.

- **Bullet:** The maximum probability of network compromise is different than the steady–state probability but this difference is limited so the usage of the method is still safe. In our analysis, we assumed the results where the ratio of maximum probability and steady–state probability was below 1.5.

We present the results of the evaluation in Table 8.7.

Table 8.7: Analyses – *compromise of network before reaching steady–state*

if the devices become	**JB**	**JLB**	**LB**	**MB**	**TB**	**Hy**
less reliable			✓	•		•
more reliable			✓	•		•
communicate more			✓	✓	•	✓
communicate less			✓	•		•
leave more		•	•			•
leave less			✓	•		•
join more		•	✓	•	•	✓
join less			✓	•		•
expand			✓	•		•
shrink			✓	•		

Dimension 3: ***Performance.***

We continue with the third dimension: *expected number of key updates after alteration.* Again we have ten rows indicating different alterations in the network dynamics, and six columns indicating different key update methods. Again we use two symbols to denote the security level after alterations, but with slightly different meanings:

- **Checkmark:** After alteration the number of key updates will not be more than it was before. There can even be less number of key updates than before. In practice, this corresponds to a negative or zero difference between the new expected number of key updates than the old one.
- **Bullet:** The alteration will cause more key updates, however the increment is limited and negligible. In our analyses, we assumed the increments less than 0.0001 per year as negligible.

We present the results of the evaluation in Table 8.8.

Table 8.8: Analyses – *performance*

if the devices become	**JB**	**JLB**	**LB**	**MB**	**TB**	**Hy**
less reliable	✓	✓	✓	✓	✓	✓
more reliable	✓	✓	✓	✓	✓	✓
communicate more	✓		✓		✓	
communicate less	✓	✓	✓	✓	✓	✓
leave more				✓	✓	
leave less	✓	✓	✓		✓	✓
join more					✓	
join less	✓	✓	✓	✓	✓	✓
expand					✓	
shrink	✓	✓	✓	✓	✓	✓

3-D: The methods satisfying all dimensions.

Finally, we combine all the results and find the intersection such that the key update methods which perform well in all dimensions. In conclusion, only a few of the strategies can survive in inly certain alterations in the network dynamics. We present the results in Table 8.9.

Table 8.9: Analyses – *intersection of all dimensions*

if the devices become	**JB**	**JLB**	**LB**	**MB**	**TB**	**Hy**
less reliable						
more reliable			✓	•		•
communicate more						
communicate less			✓			•
leave more						
leave less						
join more						
join less			✓	•		•
expand						
shrink			✓			

8.5 A Proposal of An Adaptive Key Update Mechanism for Resource-Critical Networks

In this chapter, we have shown how alteration in network dynamics can affect the performance of a network. As we have presented in our analysis, non-periodical

key update methods are very effective in preserving security in changing conditions. However, this nice adaptation in terms of security often comes with the cost of extra key updates.

Considering an extreme case that we have visualized in Fig. 8.5, when the communication in the network is altered to be more intensive, the two key update methods Hy and MB will issue a lot more key updates. The increase is so significant that can easily drain the battery of a device. As you can see in the previous chapters, in many alterations we observe increasing number of key updates in not only these two but all nonperiodical key update methods. Therefore, we claim that a comparison should consider both security and performance as we have tabulated in Table 8.9. As an example, we might be conservative on the level of security but then all the sensor devices in the network will be out of order due to excessive power consumption.

The aim of this proposal is to adapt the key update strategies to the battery level of resource-critical devices (e.g. sensor networks). As we have explained above, a key update strategy that does not consider the battery level is not very useful in the long run. Below we introduce our proposal,

First of all, we have to define *battery levels*. We assume that the trust center is able to query the battery status of the devices in the network. Trust center can receive this information as a percentage per device, and then calculate the average battery status of the network. We will divide this battery status into levels. For example, we will call *level 1* as the level where battery is either full or close to full, and *level 2* as less than half of the battery capacity available. In order to be more clear, we will define intervals that would correspond to levels.

Next, we define an *update factor* as the nonnegative real number that would control the number of key updates depending on the battery level. For example, a factor of 1 would allow the key update method to operate without any limitations, but a factor of 2 would only allow roughly a half of the key updates. Of course, we should also define a decision on *rounding* the new threshold value up or down. We can have options such as *floor* and *ceiling* functions. We can even make this choice flexible, such that for each level a different rounding can be applied.

We continue with an example where we narrate how such a mechanism can be implemented. Assume that we form a network and configure it to use a key update mechanism, with a certain key update threshold. At some point, we extended the network size by adding more devices. However, we kept the key update strategy unchanged. Of course, this caused an increase in the power consumption since we started to issue key updates more frequently to preserve the level of security as much as possible. Here comes the trade–off, if we do not

Table 8.10: Battery levels and update factors

Battery Level	Battery Interval	Update Factor	Rounding
Level 1	76% - 100%	1	None
Level 2	51% - 75%	1.25	Floor
Level 3	26% - 50%	1.5	Floor
Level 4	5% - 25%	1.75	Ceiling

employ this adaptive mechanism that we explained above, then we will have a relatively secure network but life of the network will be shorter.

Defining the battery levels and update factors in Table 8.10, we can continue our example. As you can see, we have four battery levels, and we employed generic update factors to be used with proper roundings. Using the adaptive mechanism, the first precaution will be issued when switching from level 1 to level 2. That is, when the remaining battery is less than 76% we change our update threshold to be approximately a quarter greater. Assuming that we use any of the six key update methods with the threshold values we set in this chapter, our mechanism will adjust the threshold values for each battery level as given in Table 8.11.

Table 8.11: Adapted threshold values

Method	**Level 1**	**Level 2**	**Level 3**	**Level 4**
JB	3	3	5	6
JLB	7	8	11	13
LB	3	3	5	6
MB	50	62	75	88
TB	30	37	45	53
Hy	{5,5,75,90}	{6,6,93,112}	{8,8,13,135}	{9,9,132,158}

We leave the assessment of the savings in terms of power consumption and losses in terms of security as a future work.

Part IV

Automated Tools for Analyses

Chapter 9

A Toolkit for LB Key Updates

Up until this chapter, we have presented qualitative and quantitative verification methods and developed techniques in employing them for verification of communication technologies, especially resource–critical networks such as ZigBee. We have developed models, metrics, and analyses, eventually applied on case studies. In this chapter, we intend to demonstrate some of our developments as automated tools that solve specific problems. In particular, we designed and implemented tools that can already be used by network designers and security experts. In addition, this is not a solely software development chapter since we present new developments such as building analytical model in a mathematical perspective as an alternative to what we did in Chapter 5.

In this chapter, we present a toolkit that we designed and developed that implements maximum risk analysis and transient analysis on the fluctuations in LB key update method. This toolkit is a demonstration of how to construct models in an analytical way, instead of a compositional way. Besides we have several improvements in the computations, such as eliminating redundant model checkings by terminating when maximum risk is found or when fluctuations disappeared. The toolkit can be generalized to other key update methods by replacing the analytical model construction. The toolkit is available online [Yük10].

9.1 Introduction

It is an important precaution to foresee the results of a system or a component before actually implementing it. As we have been discussing on the network systems, we have mentioned that in networking area it is often the case that dedicated or generic simulators are employed before implementing a new component such as routing protocol, collision detection method, efficiency of delivery etc. However, we encourage using formal methods that would guarantee the verification of the component such as model checking and static analysis. Based on this phenomena, we will present a simple design and development example by implementing a very useful toolkit for a specific key update strategy that can easily be used by network designers, developers, security engineers, and even ed users with a minimum knowledge. We keep the toolkit limited since this is just a demonstration of applying techniques based on formal methods in real life, especially in an area where reasoning is difficult.

In this chapter, we will present a design and implementation of a dedicated verifier that employs stochastic model checking and assists solving the problem of setting the parameters for leave based key updates in ZigBee wireless sensor networks. Even though we design the tool for a very specific purpose, it will be generic for all types of networks that use this specific key update method. We implemented the tool in MATLAB [Mat09], and benchmarked with one of the state-of-the-art stochastic model checkers: PRISM.

We are shifting from computer science perspective to mathematical perspective; namely we will build stochastic model working at a lower level, using *parameterization* to compactly specify a model. In this sense, MATLAB is a widely used programming language and tool in mathematics, since it natively supports vectors and matrices and various standard mathematical operations. Although MATLAB lacks the compositionality of higher level language-based formalisms, it is in many ways closer to the parametric specification of models preferred by mathematicians.

The purpose of designing and developing such a tool can be summarized as:

- computing the maximum risk in transient security analysis, which is not possible in generic tools
- applying a smart transient analysis which covers only the necessary time period

We can summarize the potential benefits of such a toolkit below. A *dedicated* toolkit:

- will highly increase usability, and require no expertise
- will eliminate the overhead in generic tools
- will be much faster and customized
- can also be used in benchmarking other tools

At this point we would like to mention that we limited this tool to be used in LB key update method for practical reasons, but it can be generalized to cover other methods as well. The analytical model construction stage is the mere part to be customized per method type.

In the following sections we will explain the purpose and the benefits in more details. Besides we will reuse one of our running examples, and allow comparison with PRISM.

9.2 Setting up the Scene

We are focusing on the transient behaviour of the LB key update method in the context of key confidentiality. For the sake of simplicity, we omitted the messaging between the devices. We present the related code snippet in PRISM description language in Table 9.1, slightly rearranged to increase readability.

	$NETWORK$	LB
	$Y : [0 \dots N]$ *init* N;	$Z : [0 \dots M]$ *init* 0;
	C : *bool init* $false$;	
$[join]$	$Y < N \rightarrow \lambda.(N-Y) : (Y' = Y+1)$;	$Z \leq M \rightarrow \texttt{tt}$;
$[leave]$	$Y > 0 \rightarrow \mu.Y : (Y' = Y - 1)$;	$Z < M \rightarrow (Z' = Z + 1)$;
$[leaveC]$	$Y > 0 \rightarrow \gamma.Y : (Y' = Y - 1)$ & $(C' = \texttt{tt})$;	$Z < M \rightarrow (Z' = Z + 1)$;
$[leaveR]$	$Y > 0 \rightarrow (\mu + \gamma).Y : (Y' = Y - 1)$ & $(C' = \texttt{ff})$;	$Z = M \rightarrow (Z' = 0)$;

Table 9.1: Network model with its key update environment

Now let us briefly explain the details of the model before presenting the results. We start by clearly defining the constants and variables of the model, which will lead to the full state description.

The *constants* of the model are:

- `N`: the maximum number of devices in the network
- `M`: the number of leaves required to trigger key update
- λ: the rejoin intensity
- μ: the safe leave intensity
- γ: the compromised leave intensity

The *variables* of the model are:

- $C(t)$: boolean variable that indicates if the key is compromised or not
- $Z(t)$: number of leaves since last key update
- $Y(t)$: number of active devices at time t

The full description of the *state* at time t is:

- $X(t) = (C(t), Z(t), Y(t))$

Now we can introduce the stochastic model checking results obtained from PRISM. Defining a network of maximum 20 devices, we have model checked $P_{=?}[F_{[t,t]}\ C]$ for t ranging from 0 to 3600 days with increment of 30 days each time. This corresponds to observing each month for a period of 10 years, and to increase the readability we use the letter `T` which is actually $30 * \texttt{t}$. Notice that our selection of time unit is actually in parallel with the selection of the rates we used in the model, which are specified such as (e.g. $\lambda = 1/7$,i.e. once a week. We could also get the same results if we set the time unit as a *week*, and reset the rate values accordingly. It is just a matter of design, with a condition of being used coherently.

Then comes the selection of threshold, where we used four different values ranging from 5 to 20. The values are chosen arbitrarily, yet covering a selection wide enough to observe different behaviours. It is a good practice to start with a wide range, and then narrow the threshold set until a convenient threshold is found.

The character of the network we are considering is defined with the rates as we explained before. In this example we set the rates as $\lambda = 1/7$, $\mu = 99/(365*100)$, $\gamma = 1/(365*100)$.

The resulting figure is shown in Fig. 9.1, where one can see the probability of the network key being compromised at a certain time. We plotted the result of $P_{=?}[F_{[t,t]}\ C]$ for $\mathtt{N} = 20$, $\lambda = 1/7$, $\mu = 99/(365*100)$, $\gamma = 1/(365*100)$, and $\mathtt{M} = \{5,10,15,20\}$. As you can see, the result has some interesting fluctuations which is the main reason that we chose the leave-based key update as the running example.

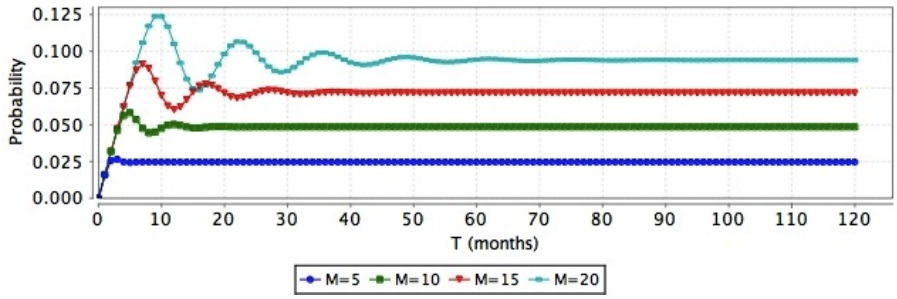


Figure 9.1: Probability of the network key being compromised at a certain time.

What we learned from this series of stochastic model checking operations is: increasing the threshold

- increases the risk that the network faces
- increases the period until reaching steady–state
- increases the quantity and also amplitude of the peaks in the result
- increases the number of model checking that is sufficient to see the behaviour of the key update method
- increases the time and memory that is needed to compute the result for a time instant

9.2.1 Problem

We now explain what is the problem we want to solve, namely what is not possible in the generic tools like PRISM in details.

In this specific key update method (as well as in similar methods such as JB, JLB, Hy, etc.) we observe fluctuations in the graphical results of transient analysis. However, current property specification formalisms are not able to

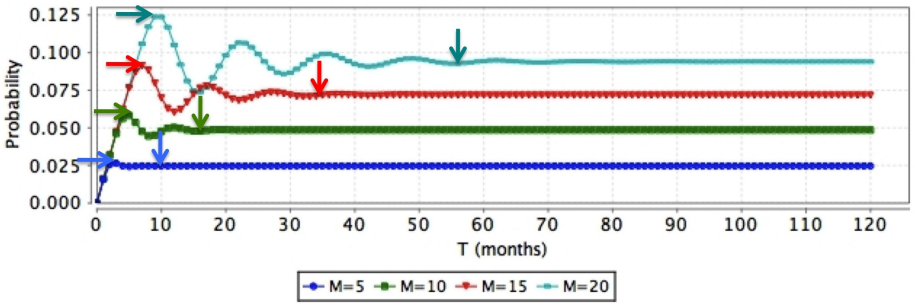


Figure 9.2: The points that we want to capture in the analysis. *Horizontal arrows*: maximum risk for a specific threshold, *Vertical arrows*: end of fluctuations for a specific threshold.

capture the peaks in the results. Thus we need to observe a sufficient time period, and manually find the time instants of peak values. In Fig. 9.2, we pointed these maximal points that we are concerned using *horizontal arrows*. For each of the four different threshold values, we have the maximum risk at a different time instant.

Another problem is, determining the period of transient analysis. In the example that we have been demonstrating, we have 120 time instants (excluding `T=0`) for each threshold. This is a safe time period, sort of an over approximation, since it covers the time before system is stabilized in terms of risk, and reached to steady–state. However, this safe approach costs 480 separate model checkings which is a lot for even such a small example. Besides, there was no guarantee that 120 time units would be sufficient per each method. Probably, it would not be sufficient if we checked for much higher threshold values. Just by looking at the graphical results, we can find out that we could save both time and memory by eliminating unnecessary model checkings and stopping model checking for a threshold roughly around the *vertical* arrows in Fig. 9.2.

9.3 Deriving the Stochastic Model

In this section, we derive the stochastic model for the CTMC in the previous section. The way PRISM constructs the model is basically taking the parallel composition of the modules written in the description language and computing the set of the reachable states from the initial state and mapping the model to the transition rate matrix.

The states of our model are encoded in a way that they reflect the values of the variables C, Z, and Y. The variable C may have the values 0 or 1 (0: the key is not compromised, 1: the key is compromised), Z may have the values from 0 to `M`, Y may have the values from 0 to `N`. We shall assume that `M`>0 and `N`>1. In our model, the key may only get compromised after a leave action, and since the model is continuous there is no need to consider multiple joins/leaves at a time.

Construction of E and Q. The first part of the model checking work is the construction of **E** (exit rates vector) and **Q** (infinitesimal generator matrix). Examining the transition system that arises from our key update model we find out that the pattern of transition is always the same. We can parametrize the system on `N` and `M` so that we can easily scale the model up and down, in other words we can exploit the regularity in the models. A very simple example of a generator matrix and corresponding state transition diagram that gives insight on the nature of the model is given in Table 9.2 and Fig. 9.3, respectively. Note that, the values for `N` and `M` are chosen small to keep the table and figure in manageable and that we wrote *rate1*=-(λ+μ+γ) and *rate2*=-2(μ+γ).

In order to ease the understanding of a conversion from a PRISM description to a generator matrix (and/or a transition state diagram), we will give examples from a very simple instance of our model where `N`=2 and `M`=2. As you can see in Table 9.1, the initial state is 002 reminding that this is the concatenation of the values of the variable C, Z, and Y. In the `NETWORK` module, all the actions are available except `join`. Mathematically, it is because the value of Y is greater than 0 but less than the value of N. Literally, it is because no new device can join when the network capacity is full. In the `ENV_leave` module, the actions `leave`, `leaveC`, and `join` are available. This case, the available functions are determined by the value of Z, namely the number of key updates after the system runs is 0. Thus, the parallel composition leads to two available actions, `leave` and `leaveC`, where the rates for the parallel actions are the same rates as in the all the actions are available except `NETWORK` module (since multiplication of the rates in the two modules will make no change, due to model design). The selection of the action is actually a race condition in PRISM, where the rates are used as parameters of the probability distribution as we mentioned earlier. Before moving our focus from the Prism model to the corresponding rate matrix, we should also explain that the resulting states after those actions will be 001 for `leave`, and 101 for `leaveC`.

Now we can view the same case in the transition rate matrix and the transition state diagram, presented in Table 9.2 and Fig. 9.3, respectively. In fact, we will use the generator matrix instead of the rate matrix where the only difference is in the diagonals, namely no transition to another state. In fact, from now on we will only refer to the *infinitesimal generator matrix* **Q** in order to avoid

confusion. Since we have chosen a simple instance of the model, the diagram is easier to generate and understand. Notice that for the sake of simplicity we omitted all the self loops which should be present in all the states. The initial state is now easily identified by the incoming transition arrow that does not originate from any of the other states. Then you can see two outgoing transitions as dotted arrows, and the corresponding rates on top of te arrows. This, actually is the same transitions we explained in the previous paragraph, and in the matrix representation you can see the originating states in the rows (the third row in this case) and the resulting states in the columns (the fifth and the eleventh in this case, excluding the self loops of course).

Table 9.2: Infinitesimal Generator Matrix **Q** for N=2 and M=2

CZY	000	001	002	010	011	012	020	021	022	110	111	112	120	121	122
000	-2λ	2λ													
001		rate1	λ	μ						γ					
002			rate2		2μ						2γ				
010				-2λ	2λ										
011					rate1	λ	μ						γ		
012						rate2		2μ						2γ	
020							-2λ	2λ							
021	μ+γ							rate1	λ						
022		-rate2							rate2						
110										-2λ	2λ				
111											rate1	λ	μ+γ		
112												rate2		-rate2	
120													-2λ	2λ	
121	μ+γ													rate1	λ
122		-rate2													rate2

Obviously, we want to construct the **Q** matrix which is convenient for computations, and also **E** vector which will be used in this construction (and also later computations).

We start our computations with **E**, which is the vector of exit rates for all the states. It is trivial by inspecting the matrices or the transisiton diagrams that, the number of states can be parametrized by N and M such that there are (M+1)*(N+1)*2 states depending on the possible values of the variables, however the states where C=1 and Z=0 are not reachable so we have $(2\mathtt{M}+1)(\mathtt{N}+1)$ states in total. Actually, **E** can be constructed by vertically concatenating 2M+1 identical little vectors ($\mathbf{E_0}$). Since $\mathbf{E_0}$ has N+1 rows, **E** will have (2M+1)(N+1) rows.

We derive the formulation of the terms of the $\mathbf{E_0}$ vector as:

$$\mathbf{E_0}(i) = (\mathtt{N}-1).\lambda + i.(\mu+\gamma) \quad i = 0, .., \mathtt{N}$$

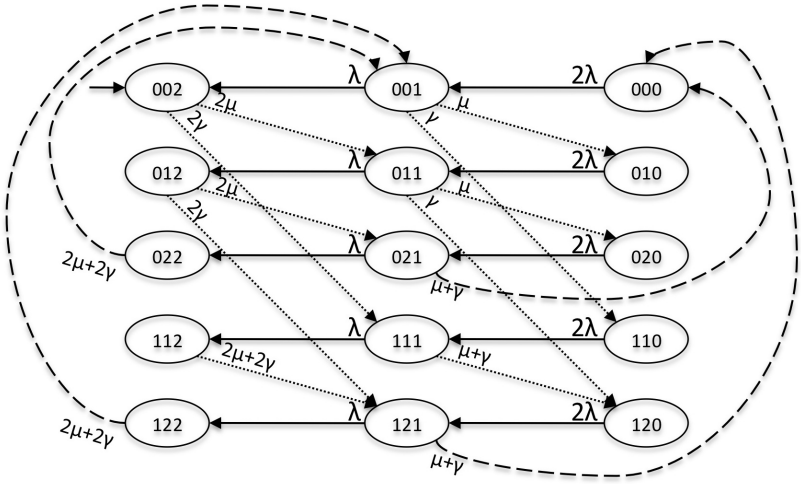


Figure 9.3: State-Transition Diagram for N=2 and M=2

Using $\mathbf{E_0}$, we can easily construct the $\mathbf{E}$ vector as:

$$\mathbf{E} = \begin{bmatrix} \mathbf{E_0} \\ \vdots \\ \mathbf{E_0} \end{bmatrix}$$

After computing the $\mathbf{E}$ vector which will also be used in uniformisation to be explained later, we can start describing a smart way of constructing the $\mathbf{Q}$ matrix. By inspecting the instances for the $\mathbf{Q}$ matrix, we perform a divide and conquer approach where we first deal with the submatrices. Below are the four matrices that we name as A_λ, A_μ, A_γ, and $A_{\mu\gamma}$ (for the sake of readability we omitted the zeros in the matrices):

$$\mathbf{A}_\lambda = \begin{bmatrix} -\mathbf{E}(\mathbf{1}) & \mathtt{N}.\lambda & & & & \\ & -\mathbf{E}(\mathbf{2}) & (\mathtt{N}-1).\lambda & & & \\ & & \ddots & \ddots & & \\ & & & -\mathbf{E}(\mathtt{N}) & 1.\lambda & \\ & & & & -\mathbf{E}(\mathtt{N}+\mathbf{1}) & 0.\lambda \end{bmatrix}$$

$$\mathbf{A}_{\mu} = \begin{bmatrix} 1.\mu & & \\ & \ddots & \\ & & \mathtt{N}.\mu \end{bmatrix} \quad \mathbf{A}_{\gamma} = \begin{bmatrix} 1.\gamma & & \\ & \ddots & \\ & & \mathtt{N}.\gamma \end{bmatrix}$$

$$\mathbf{A}_{\mu\gamma} = \begin{bmatrix} 1.(\mu+\gamma) & & \\ & \ddots & \\ & & \mathtt{N}.(\mu+\gamma) \end{bmatrix}$$

Using the similarity in the matrices $\mathbf{A}_{\mu}$, $\mathbf{A}_{\gamma}$ and $\mathbf{A}_{\mu\gamma}$, we come up with a single matrix as a substitute, $\mathbf{A_u}$:

$$\mathbf{A_u} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & \mathtt{N} \end{bmatrix}$$

The relation between $\mathbf{A_u}$ and the matrices that it substitutes can be defined as:

$$\mathbf{A}_{\mu} = \mu.\mathbf{A}_u \tag{9.1}$$

$$\mathbf{A}_{\gamma} = \gamma.\mathbf{A}_u \tag{9.2}$$

$$\mathbf{A}_{\mu\gamma} = (\mu+\gamma).\mathbf{A}_u \tag{9.3}$$

Now we can rebuild our $\mathbf{Q}$ matrix by using the little matrices above:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \\ \mathbf{Q}_3 & \mathbf{Q}_4 \end{bmatrix}$$

$$\mathbf{Q_1} = \begin{bmatrix} \mathbf{A}_{\lambda} & \mu.\mathbf{A}_u & & \\ & \ddots & \ddots & \\ & & \mathbf{A}_{\lambda} & \mu.\mathbf{A}_u \\ (\mu+\gamma).\mathbf{A}_u & & & \mathbf{A}_{\lambda} \end{bmatrix} \quad \mathbf{Q_2} = \begin{bmatrix} \gamma.\mathbf{A}_u & & \\ & \ddots & \\ & & \gamma.\mathbf{A}_u \end{bmatrix}$$

$$\mathbf{Q_3} = \begin{bmatrix} & \\ (\mu+\gamma).\mathbf{A}_u & \end{bmatrix} \quad \mathbf{Q_4} = \begin{bmatrix} \mathbf{A}_\lambda & (\mu+\gamma).\mathbf{A}_u & & \\ & \ddots & \ddots & \\ & & \mathbf{A}_\lambda & (\mu+\gamma).\mathbf{A}_u \\ (\mu+\gamma).\mathbf{A}_u & & & \mathbf{A}_\lambda \end{bmatrix}$$

At this point, we can state the dimensions of the matrices that we use in the construction of $\mathbf{Q}$, such that

- $((m+1).(n+1)) \times ((m+1).(n+1))$ for $\mathbf{Q}_1$,
- $((m+1).(n+1)) \times (m.(n+1))$ for $\mathbf{Q}_2$,
- $(m.(n+1)) \times ((m+1).(n+1))$ for $\mathbf{Q}_3$, and
- $(m.(n+1)) \times (m.(n+1))$ for $\mathbf{Q}_4$

Thus we have presented our first contribution in constructing the generator matrix. We implemented all these constructions in MATLAB (see Appendix E).

9.4 Model Checking Computations

As we have explained in the preliminaries chapter, stochastic model checking of a CTMC model requires a conversion to an *embedded DTMC* in order to obtain (more) numerically stable computations and to avoid round-off errors that are caused by the positive and negative values in $\mathbf{Q}$.

An important issue is **uniformisation**, a technique that is used for computing transient probabilities of CTMCs and relied on in the CSL model checking algorithms.

As we have introduced earlier, the transition probability matrix for the uniformised DTMC as $\mathbf{P}^{unif} = \mathbf{I} + \mathbf{Q}/q$ such that the *uniformisation rate* $q \geq max\{E(s)|s \in S\}$. PRISM implements q as $1.02 * max\{E(s)|s \in S\}$, and in order to be able to benchmark our results we also use this calculation[1].

At this point, we need to remind a matrix that we will use in our computations. The matrix of all transient probabilities for time t was defined as $\mathbf{\Pi_t}$, and can be

[1]The usual approach is to take $q = max(E(s))$, which is also observed in some other probabilistic model checkers (e.g. MRMC) [Ste94, BHHK00].

expressed as a matrix exponential such that $\mathbf{\Pi_t} = e^{\mathbf{Q}.t}$. However, computing this (as power series) tends to be unstable, therefore the probabilities are computed through the uniformised DTMC $\mathbf{P}^{unif}$ instead of CTMC. This brings us to the equality:

$$\mathbf{\Pi_t} = \sum_{i=0}^{\infty} \gamma_{i,qt} \quad (\mathbf{P}^{unif})^i \quad \text{where}\, \gamma_{i,qt} = e^{-qt}\frac{(qt)^i}{i!} \tag{9.4}$$

where $(\mathbf{P}^{unif})^i$ is the probability of jumping between each pair of states in i steps, and $\gamma_{i,qt}$ is the (Poisson) probability of i such steps occurring in time t, given that the delay is exponentially distributed with rate q. Note that, (unlike $\mathbf{Q}$) the matrix $\mathbf{P}^{unif}$ is **stochastic**, *i.e.* all entries are in the range [0,1] and all rows sum to 1.

After introducing the necessary conversions and computations, we can continue with the core part of CTMC model checking.

The logic formula $P_{=?}[F_{[t,t]}\, C]$ can be rewritten as an *until* formula as $P_{=?}[true\, \mathrm{U}^{[t,t]} C]$. This enables us to determine the probability using the formula from [KNP07]:

$$\underline{Prob}^{\mathcal{C}}(\Phi\ \mathrm{U}^{[t,t']}\ \Psi) = \sum_{i=0}^{\infty} \left(\gamma_{i,qt} \cdot (\mathbf{P}^{unif(\mathcal{C}[\neg\Phi])})^i \cdot \underline{Prob}^{\mathcal{C}}_{\Phi}(\Phi\ \mathrm{U}^{[0,t'-t]}\ \Psi)\right) \tag{9.5}$$

where obviously Φ is $true$, Ψ is C, and t is equal to t' in our case. Inside the infinite summation we have three parts to be multiplied, the first one of which, , namely $\gamma_{i,qt}$, was explained in the formula (9.4). The second part is the ith power of the $\mathbf{P}^{unif}$ of the CTMC $\mathcal{C}[\neg\Phi]$, that is $\mathcal{C}[false]$ (note that C is the variable that indicates whether the key is compromised, whereas $\mathcal{C}$ is the CTMC). At this point, we should remind the reader that for any CTMC $\mathcal{C}$ and CSL formula Φ, CTMC $\mathcal{C}[\Phi]$ is obtained by replacing $\mathbf{R}$ by $\mathbf{R}[\Phi]$ such that $\mathbf{R}[\Phi](s, s0) = \mathbf{R}(s, s0)$ if $s \not\models \Phi$ and 0 otherwise.

Now we instantiate this equation with the details that are specific to the ZigBee key update model. It is trivial to derive that the last part is actually a vector for Ψ.

Integrating the equations above, we get our specific equation that we need to

solve in order to get the model checking results, that is:

$$\underline{Prob^{\mathcal{C}}}(true\ \mathrm{U}^{[t,t]}\ C) = \sum_{i=0}^{\infty} \left(e^{-qt} \frac{(qt)^i}{i!} \cdot (\mathbf{I} + \mathbf{Q}/q)^i \cdot \underline{C} \right) \tag{9.6}$$

Improving the model checking time. Here we have another improvement that occurs from engineering the process. Instead of going through the complete model checking formula and applying the substitutions, we directly use the simplified formula in (9.6) and gain time.

In addition, we have a considerable improvement in the model checking time for the type of problem in this running example. Even though PRISM allows computing a series of model checkings and produce the result in graphical form (and of course numerical data as well) which is called *experiments* in PRISM jargon, it computes the uniformisation and the Q matrix everytime from scratch. Since the uniformisation and the generator (or rate) matrix is always the same for all model checkings in an experiment, doing it only once is a considerable improvement in model checking time.

We should note that our implementation is based on sparse matrices, therefore competing with the sparse matrice engine of PRISM. In fact, PRISM is a *symbolic* model checker that also uses data structures based on binary decision diagrams (BDD). This feature of PRISM provides compact representation and efficient manipulation of probabilistic models by exploiting regularity.

9.5 Specific Technicalities in Computation

In the previous section we presented the formula to be computed in order to get stochastic model checking results. However, solving that formula is not always straightforward and may require mathematical tricks. In this section we will explain some key points of the successful implementations.

One hard problem we faced in the implementation was the infinite summation. Before inspecting how the existing model checkers deal with the problem, we figured out that the summation has a specific pattern that is a sequence of three phases such that the first and the last phases have negligible values. Therefore we chose a naive way of skipping the computation in these phases where the probabilities were insignificant. After that we realized that we were close to rediscover a well-known result dating more than twenty years back: the Fox/Glynn

Method [FG88]. This technique actually allows efficient computation of Poisson probabilites (i.e. the first part of the infinite summation in equation (9.6)) and furthermore produces upper and lower bounds for the point where below and above the probabilities are insignificant. This operation is also known as truncation of the infinite summation and the bounds are called the left and right truncation points. As a result, the number of the iterations that are necessary to compute the probabilities are supplied by the Fox/Glynn Method.

An interesting fact that is revealed by most of the designers of such tools is iterative squaring is not attractive for sparse matrices due to fill-in.

9.6 Design of the Toolkit

We illustrate the structure of the toolkit in Fig. 9.4. Each colored box represent a different tool, blue ones being the *front end* tools and purple ones being the *back end* tools. The front end tools can be used with the inputs given on top of them, which is a network specification including size and rates, and an LB strategy specification including a threshold or a threshold set. These tools return results of transient analysis and the maximum risk that the network can face. The back end tools are implementing the stochastic model checking for the desired property. One of them constructs the necessary matrices and sets the variables to be used in model checking, and the other actually implements model checking. Back end tools can also be used by expert users to experiment various scenarios, and compute model checking results.

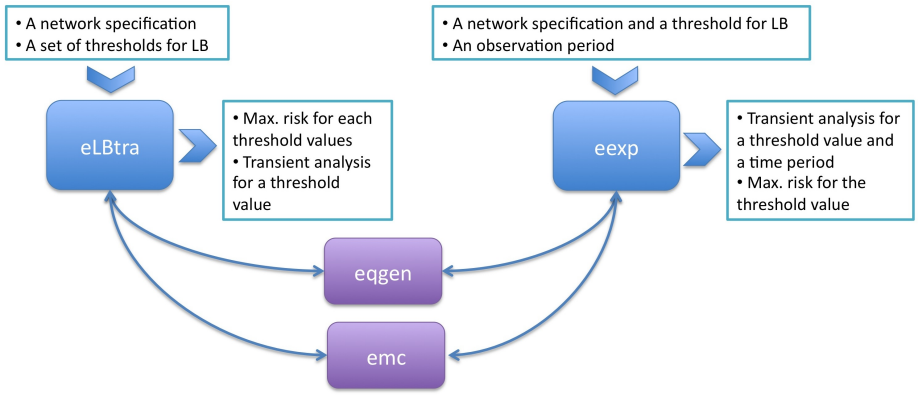


Figure 9.4: The structure of the toolkit.

In the following two sections we present the components of the toolkit, namely

the tools we designed.

9.6.1 Front End Tools

We have two different front end tools, **eexp** and **eLBtra**, both focusing on assisting the user to calculate the risk for a scenario. This two tools have different perspectives that we explain below.

eexp(`N`, `M`, λ, μ, γ, t_0, t_1, [SC], [SCpar]): model checks the CSL Time-bounded Until formula for the ZigBee Leave-based key update method. This tool is for expert users, reveals more information on the computations and allows more detailed adjustments in the whole process.

- computes number of states and matrix sizes automatically, without constructing the actual state space.
- constructs the CSL formula as a vector, $\mathbf{B}$.
- computes $\mathbf{A}$ ($\mathbf{P}^{unif}$) and q by calling **eqgen**.
- computes model checking for time instants between t_0 and t_1 by calling **emc** as many times as needed.
- depending on the [optional] stopping criteria switch and parameter, allows choosing a stopping criteria between
  1. **[SC = 0]** limited number of iterations, SCpar being the limit (similar to `TERMINATION_MAX_ITERATIONS` in PRISM)
  2. **[SC = 1]** convergence, SCpar being the epsilon (similar to `TERMINATION_EPSILON` in PRISM)
- computes the elapsed times of:
  1. $\mathbf{E}$, q, $\mathbf{Q}$, $\mathbf{A}$, $\mathbf{B}$ generation
  2. Time bounded CSL Until Model Checking

`Returns:` risk of key compromise for

- a network specified with maximum size `N`, and rates λ, μ, γ,
- employing LB key update strategy with a threshold of `M` devices,
- in the time period between t_0 and t_1.

eLBtra(N, $\mathtt{M}_{init}$, $\mathtt{M}_{step}$, $\mathtt{M}_{end}$, λ, γ): Automated tool for computing maximum risk in LB Key Update. This tool is the main tool in the toolkit, and can be used by end users as well as the experts. Below we summarize the features that this tool differs from the expert tool **eexp**.

- **eLBtra** computes results for a set of thresholds (starting with $\mathtt{M}_{init}$, incrementing with step $\mathtt{M}_{step}$, until reaching $\mathtt{M}_{end}$), whereas **eexp** works on a specific threshold (M).
- **eLBtra** has fewer input parameters in order to make it easier to use (eliminating time inputs, safe leave intensity, and optional stopping criteria).
- **eLBtra** does not output any unnecessary detail for an end user.
- instead of entering the time period for stochastic model checking, **eLBtra** finds the point where the results are close enough to the steady–state. Therefore, it saves time and memory, and avoids trials for finding the correct time period.
- if the user is only interested in the maximum risk, then **eLBtra** stops the stochastic model checking sequences when the peak value is reached (exploiting the nature of LB key update method, and in general the methods where we observe fluctuations).
- uses the convergence method as the stopping criteria.

9.6.2 Back End Tools

We have two different back end tools, **eqgen** and **emc**, that will compute the desired results when called sequentially. We also have an auxiliary tool **foxglynn** which is not our design but our implementation of a well known method, Fox-Glynn.

eqgen(N, M, λ, μ, γ): Constructs the infinitesmal generator matrix for the leave-based key update scenario, focusing on efficiency. Called by **eexp** or **eLBtra**.

- computes $\mathbf{E}$ (by first computing $\mathbf{E_0}$).
- computes auxiliary matrices for $\mathbf{Q}$ ($\mathbf{A_u}$, $\mathbf{A}_\gamma$, $\mathbf{A}_\lambda$, $\mathbf{A}_\mu$).
- computes $\mathbf{Q}$ (by first computing the quadrants $\mathbf{Q_1}$, $\mathbf{Q_2}$, $\mathbf{Q_3}$, $\mathbf{Q_4}$).
- computes uniformisation rate q, and probability matrix for the uniformised DTMC $\mathbf{P}^{unif}$.

`Returns:` q and $\mathbf{P}^{unif}$ (renamed as $\mathbf{A}$ in parameter lists).

emc($\mathbf{A}$, $\mathbf{B}$, q, t, SC, SCpar): Computes model checking result for a specific time instant. Called by **eexp** or **eLBtra**.

- implements model checking for time bounded CSL Until formula $P_{=?}[true\ \mathrm{U}^{[t,t]}\ C]$, which amounts to $P_{=?}[F_{[t,t]}\ C]$.
- depending on the *stopping criteria* switch (SC), uses either a limited number of iterations or truncated summation assisted by Fox-Glynn method (calling **foxglynn**).

`Returns:` stochastic model checking result for $P_{=?}[F_{[t,t]}\ C]$.

foxglynn(qt_{max}, underflow, overflow, accuracy): Computes poisson probabilities for uniformisation, implementing Fox-Glynn method. Based on the PRISM implementation by Joachim Meyer-Kayser. Called by **emc**.

- computes left and right truncation points for the computation of poisson probabilities.
- to allow benchmarking with PRISM, we set precisely the same values to underflow, overflow, and accuracy parameter.

`Returns:` truncation points computed by Fox-Glynn method.

9.7 Demonstration

In this section, we demonstrate the usage of the front end tools. The complete code listings are provided in Appendix E.

Tool eexp. We start by using the tool **eexp** for a small set of values. We investigate the security of a network of maximum two devices, and LB threshold of two devices. This is exactly the same model that we used extensively in Section 5.3.3. We present the output of the tool in Fig. 9.5, where we input (2,2,0,0,0,1,5) that is {2,2} for `N` and `M`, {0,0,0} to use the default values for rates (i.e. $\lambda = 1/7$, $\mu = 99/(365*100)$, $\gamma = 1/(365*100)$), and {1,5} to issue model checking at five consecutive time instants starting from `t=1`.

We observe exactly the same value for the uniformisation factor as in PRISM, which is no surprise because we are implementing the same algorithm. However, we compute this uniformisation factor and all the model construction only once

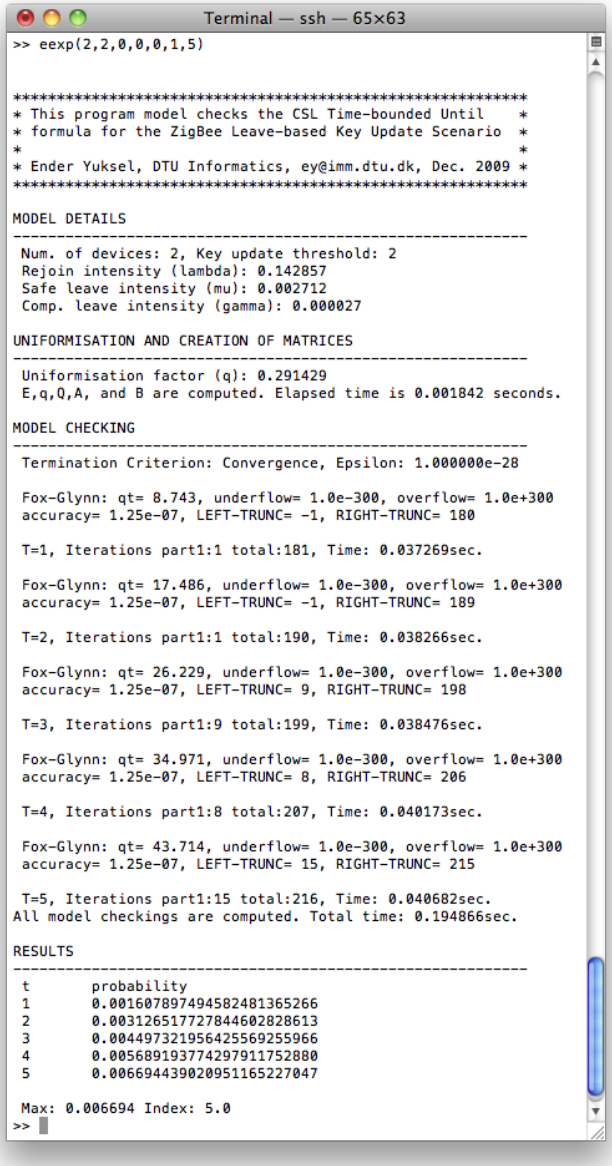


Figure 9.5: Computing the risk in the first five time units for `N=2`, `M=2`.

whereas PRISM repeats the computation for each time point. Even for computing once, our MATLAB model is much faster since we exploit the stochastics by constructing the generator matrix using our analytical method that we developed in Section 9.3. Of course we also compute the same truncation points from Fox-Glynn (excluding the shifting in our indexing), and eventually we get the same probability results. At the end, we also present the maximum risk for the time period given in input.

Unfortunately the server that we run MATLAB was not as powerful as the personal computer that we ran PRISM, which prevented us from making a comparison. Besides as a programming language MATLAB is much slower than C++ which is the language that PRISM engine is coded with. As a result, we cannot claim being faster in model checking computations.

Tool eLBtra – ***maximum risk analysis.*** Our next example will be using the *user–friendly* front end tool **eLBtra** which requires minimum knowledge to operate and sheds light on a wider series of questions: *which threshold has which maximum risk, at which time instant, and when is the risk reaching steady–state.* In Fig. 9.6, we present a simple run of **eLBtra** for a network of up to 20 devices, exactly the same example that we explained in Section 9.2. Using the tool we can find the maximum risk for a set of threshold values, together with the time instants for the maximum risk. Thus, we instead of having a long series of model checking we can easily foresee what will a network come up against as highest risk, and when will be this critical period. In this example, **eLBtra** only computes 28 model checkings in total which is much less compared to 480 model checkings in Fig. 9.1. We will explain the logic behind this improvement in details below, in Section 9.7.1.

Tool eLBtra – ***transient analysis covering all fluctuations.*** After getting the maximum risk for a certain set of threshold values, we can choose a threshold value and let **eLBtra** continue the transient analysis until a steady–state is reached. Doing so, we can get more insight on the fluctuations in the risk and exploit the tools capability of automatically stopping when the results are sufficiently close — determined by a convergence epsilon — to a steady–state. In Fig. 9.7, we present the output for the transient analysis of `M=5` where the analysis continues until `t=16`.

In addition, we may ask **eLBtra** to compute results for a single threshold value, rather than a set of values. Continuing with our example where we explained the problem, for the threshold value of 15 **eLBtra** computes results for 33 time instants and outputs that these model checkings are sufficient to fully observe the behaviour of the network. Hence, again we saved time and memory compared to our results in PRISM. The corresponding output of **eLBtra** is presented in Fig. 9.8.

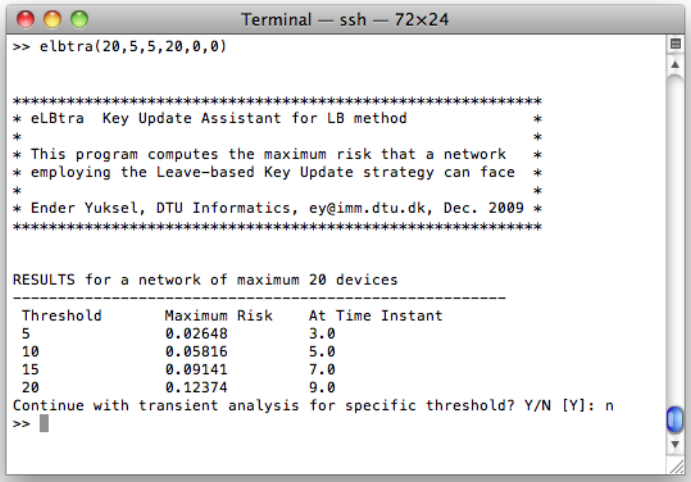


Figure 9.6: Computing the maximum risk for different threshold values.

9.7.1 Discussion on the termination of the model checkings

In this section, we give a more concrete answer on how we found a solution for the problems that we explained in Section 9.2.1 and visualized in Fig. 9.2. As we designed a dedicated tool for a specific key update method, we can perfectly exploit the characteristics of the method (which can actually be reused in similar methods with a slight modification). As we have explained earlier, we constructed the generator with the same approach and improved the computation of the model. Now we explain how we benefit in number of model checkings based on our front end tool **eLBtra**.

In Fig. 9.9, we reissue the results that we presented when setting the scene and defining the problem focusing on the maximum risk. In this reissue, we highlighted the points that **eLBtra** computed to find the maximum risk for each threshold and shaded the rest of the points. In LB method, the risk monotonically increases until each reaches the first local maximum which is also the global maximum. It is simple to check if we reached the global maximum by comparing two consecutive values. For instance, taking the case `M=15` where we listed the probabilistic values in Fig. 9.8 the tool will start computing from `t=1` (the first highlighted point in the figure) and when `t=8` (the last highlighted

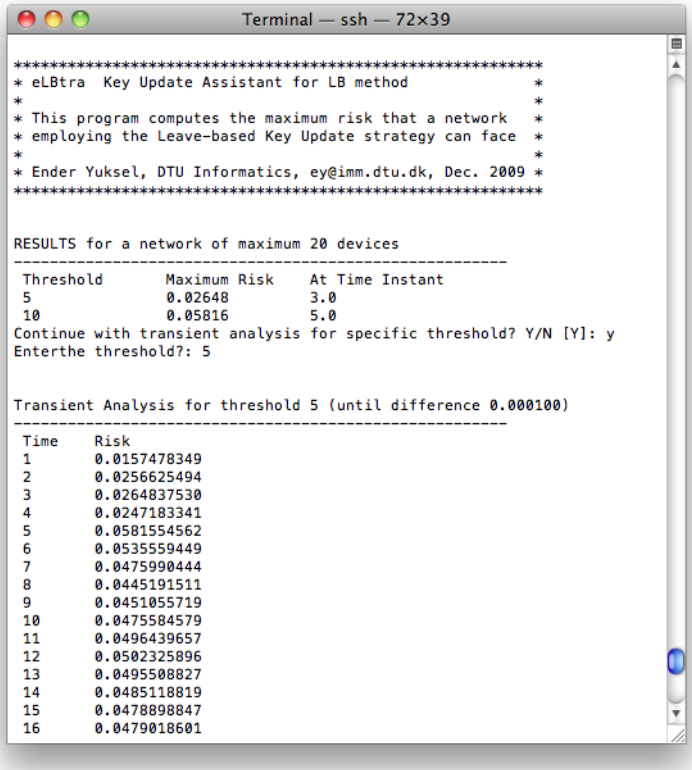


Figure 9.7: Transient analysis for `M=5`.

point in the figure) it will find out that the difference with the predecessor is negative and stop computations.

We follow a similar approach in the transient analysis to find the time instant to stop computations. In Fig. 9.10, we present another reissue of the results where the points of transient analysis in **eLBtra** is highlighted. This time we are checking the *absolute* difference between the consecutive probabilistic results. We have a predefined convergence epsilon, that can easily be adjusted in the cases where precision is crucial, and when the difference is no more greater than this epsilon value the computation is terminated. In Fig. 9.10 we presented the necessary computations that are issued by **eLBtra**.

As **eLBtra** initially computes the maximum risk, we reuse the points that are already computed there and continue the transient analysis with the remaining

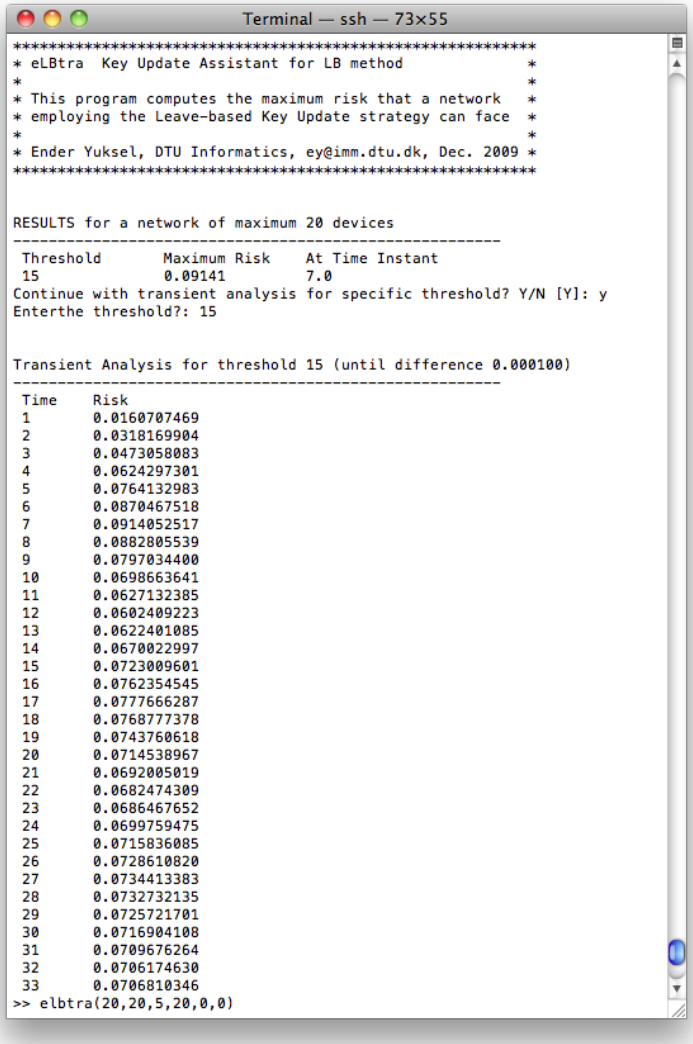


Figure 9.8: Transient analysis for `M=15`.

points. As a solid example, for the case of `M=15` in Fig. 9.10, the points from `t=1` to `t=8` are already computed for the maximum risk, therefore if we request a transient analysis covering all the fluctuation then **eLBtra** only computes the remaining points until `t=33`.

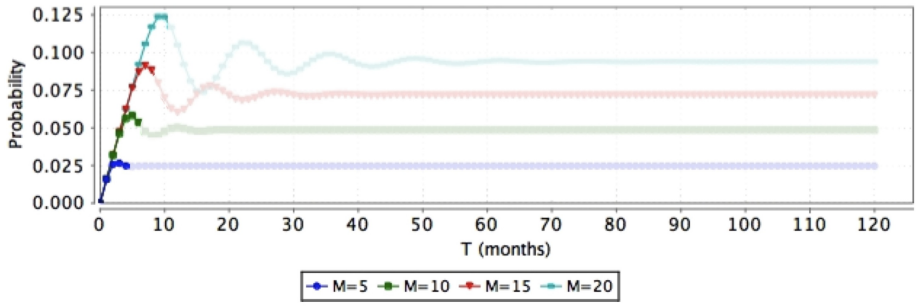


Figure 9.9: Analysing the maximum risk in different thresholds.

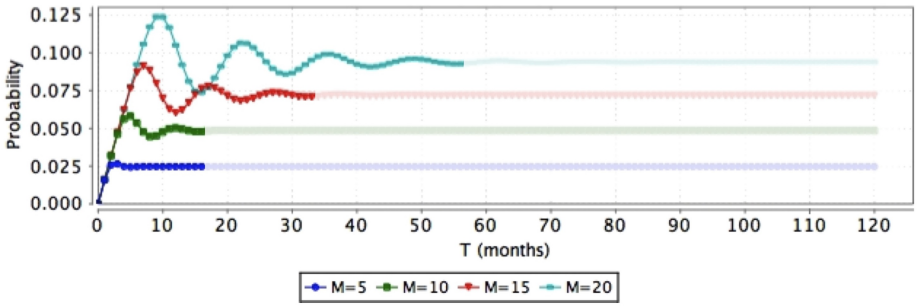


Figure 9.10: Analysing the period before reaching steady–state.

Chapter 10

Automated Tools Utilizing Present Technologies

In this chapter, we present automated tools that we designed and developed to solve some of the problems we discussed throughout the thesis. Once again we provide solutions in an automated manner and eliminate the necessity of an expertise for the users of the tools. As a difference from the previous chapter, we intended to employ state-of-the-art verification tools which require expertise to use.

In Section 10.1, we designed and developed a software that automates the decision of key update method and parameters for a specific network. We created a *push-button* technology – powered by stochastic model checking – that network designers and even consumers can easily benefit from. *Key Update Assistant*, as we named it, runs necessary model checking operations and determines the optimum key update strategy that satisfies the requirements of its users.

In Section 10.2, we designed and developed a protocol verifier that bridges the gap between protocol narrations and LySa models, and also automates qualitative verification using LySa engine. This tool allows verification of a protocol in various encapsulation forms involving integrity and encryption as in IEEE 802.15.4 and ZigBee.

Both of the tools presented in this chapter are available online [Yük10].

10.1 Automating the Decision on Key Update: *Key Update Assistant*

In this section, we design and develop a software that automates the decision of the key update method and threshold parameter. Our aim is to create a *push-button* technology that will find the optimum key update strategy – formed by a *method* and an *update threshold* – for network designers and consumers by verifying security and performance properties of a custom network.

We will make use of our developments in the quantitative analysis domain and provide solution for any specific network design. We start by presenting a rather shallow flowchart of *Key Update Assistant* – the program that we designed – in Fig. 10.1.

Key Update Assistant – in a blackbox perspective – takes the requirements of the user as input, and returns the satisfying key update strategy and key update threshold. In the background, the program runs a probabilistic model checker (abbreviated as *MC* in the flowchart) to verify the configurations and to find the optimum solution. This process may require refining the set of thresholds sent to the model checker and repeating the middle steps.

In the following sections, we will explain each box of the flowchart in Fig. 10.1. Section 10.1.1 contains details on the first box, Section 10.1.2 contains the inner loop namely the computation of the threshold values, Section 10.1.3 is on the involvement of probabilistic model checking engine, and Section 10.1.4 contains details on the outer loop, namely deciding on the solution. Finally, in Section 10.1.5 we present a little demonstration of the tool.

10.1.1 Input

The input to the program consists of three types of information:

- security and performance requirements
- network parameters, and
- key update methods

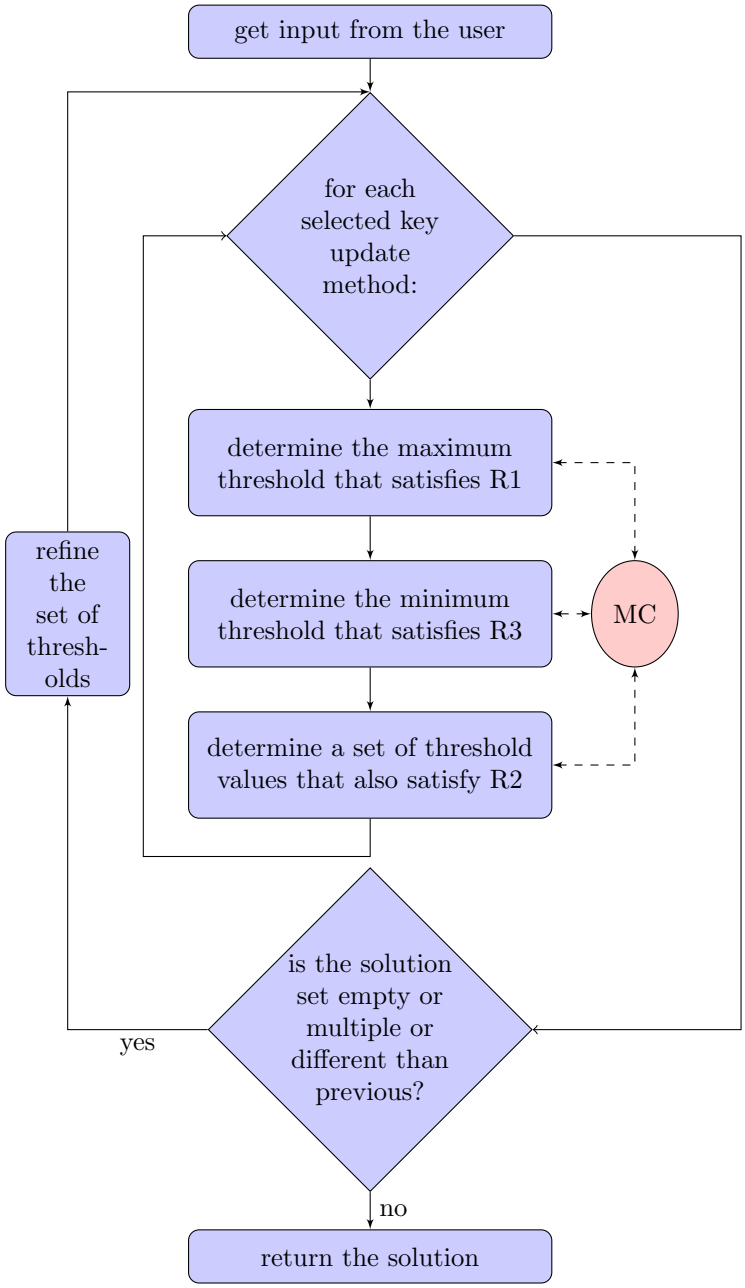


Figure 10.1: Flowchart of *Key Update Assistant*

Security and performance requirements are the main requirements that would identify the what is requested from the network in terms of security and performance guarantees. For the sake of simplicity, we restrict this part of input with three fields (*all fields are necessary*):

R1 Maximum tolerated probability of the network key being compromised in the long run.

R2 Maximum tolerated probability of the network key being compromised in any time instant.

R3 Maximum tolerated number of key updates in a year

Network parameters describe the characteristics of the network and application scenario. This information helps in reasoning about how dynamic/stationary is the network, how frequently the devices in the network communicate, how safe is the environment, etc. Again, for the sake of simplicity we restrict this part of input with two options (*choose one out of two options*):

O1 Select one of the six ZigBee Application Profiles, or

O2 Customize your network scenario by entering the parameters below:

- **O2.a** Rate of join per device
- **O2.b** Rate of leave per device
- **O2.c** Rate of message per device
- **O2.d** Probability of compromising the key when leaving per device
- **O2.e** Probability of compromising the key when messaging per device
- **O2.f** Initial size of the network
- **O2.g** Maximum size of the network

Key update methods define allowed key update methods for the network in consideration. This input is limited by the models we have in the program, however new key update methods can be added afterwards. We restrict this part of input with six options (*multiple choice*):

Hy Hybrid key update

JB Join-based key update

JLB Join-Leave-based key update

LB Leave-based key update

MB Message-based key update

TB Time-based key update

After the input is taken from the user correctly, formal models of the key update methods will be configured to reflect the desired network settings and will be ready for model checking. Note that property specification is always the same since the requirement questions (R1, R2, and R3) are constant.

10.1.2 Identifying The Thresholds

The three consecutive boxes in the inner loop are the core parts of the software. In this section, we will present the logic behind those modules.

The first box is identifying the *maximum* threshold for a certain key update method according to R1 requirement. Here we will take a refinement strategy to determine maximum threshold value for each selected key update method. We designed this strategy to avoid redundant and resource consuming model checking operations. We present our methodology for doing so in Table 10.1.

Basically we are exploiting the fact that, the difference between steady–state probabilities for successive threshold values would be decreasing but still useful for skipping many model checking operations. As an example, instead of model checking for 100 different threshold values starting from 2 to 101, we can compute first two and then jump to a threshold value that is close to 101 (depending on the rest of the parameters) and then continue iterating by successively incrementing the threshold value. Thus we can omit a huge amount of model checking which would save us time and memory.

The second box is determining the *minimum* key update threshold according to R3 requirement. Making use of our results in the previous box, namely the maximum threshold, we iterate through threshold values starting from maximum and decrementing by one in each iteration. Obviously, the iteration does not start if the model already violates R3. A selected key update method will be eliminated if no threshold values satisfies R1 in the previous stage, or no threshold values satisfies R3 in this stage. After successful computation of the second box we have a maximum threshold MAX and a minimum threshold MIN.

STEP I: Compute the probability difference between two successive threshold values:

1.a) Compute steady-state probability for threshold value $thr = 2$, and save the result as ss_2
1.b) Increment the threshold by one (i.e. $thr = 3$) and compute steady-state probability and save the result as ss_3

STEP II: Find the offset value to start model checking iterations:

2.a) Compute the value of $offset$ as:

$$offset = (int)\frac{R1 - ss_3}{ss_3 - ss_2}$$

where $R1$ is the first requirement
2.b) Compute steady-state probability for threshold value $thr = 4 + offset$, and save as ss_{thr}

STEP III: Start model checking iterations by gradually adjusting the threshold value:

3.a) Set $difference$ as $diff = R1 - ss_{thr}$
3.b) while $|diff| > convergence\_epsilon$ set threshold as:

$$new\_threshold = old\_threshold \mp \frac{old\_threshold}{2}$$

$\mp$ being $+$ if $diff$ is positive, and $-$ otherwise.
Compute the result for the new threshold, and recompute $diff$

STEP IV: Return the result that either converged to a specified epsilon value or a threshold increment that is no more meaningful (i.e. less than 1):

4.a) If the value of $diff$ is non-negative then return the current threshold value
4.b) Otherwise (i.e. $diff < 0$), decrement threshold by one and return it

Table 10.1: Methodology for determining maximum threshold value depending on requirement R1

The third box actually produces a solution set by making use of previous three stages. Starting from MIN and ending in MAX, all the threshold values are actually a solution for the given problem. Surely, the minimal part of this set

will serve as a better solution in security perspective and also will not violate performance requirements. Therefore, we limit the number of threshold values in solution set as 10.

10.1.3 Model Checker

In principle, any probabilistic model checker would work in this program provided that we supply the model formalized in the model checkers description language or input method. So to speak, MRMC [KKZ05], E⊢MC^2 [HKMKS00], VESTA [SVA04], YMER [You05] are some of the alternatives one can choose. In our case, we will stick to PRISM probabilistic model checker that we explained and used in this thesis. We have provided the models and property specification in the Appendix C that can directly be used for this program.

10.1.4 Deciding On The Solution

In the outer loop of the flowchart, we consider the *solution* which is a collection of the solution sets – threshold sets – for each selected key update method. If we have one unique pair of solution such that a pair p is a tuple (ku, th) where ku is the key update method and th is the threshold for that method, then we are happy and we return the solution as the output of the program. Else if we have no solutions, that we have to refine the threshold set to the model checker by coarsening. Else if we have more than one pairs in the solution, then we refine the threshold set by tightening. We terminate the tightening if we do not get better results.

10.1.5 Demonstration

In this section, we present a demonstration of the tool we implemented. Above we explained the design of the tool and now we will add technical details on the implementation. We implemented the Key Update Assistant in Java, and tested in Mac OS X platform. The tool has a graphical user interface, that allows its user to easily enter the input and see the results. In the background, the tool runs probabilistic model checker which is PRISM in this case. The user does not need to know about how many times and with which arguments the model checker engine is called, or how the properties are expressed. However, necessary information is produced and presented in the Details section of the

interface. All the implementation actually complies with our developments and algorithms in the previous chapters.

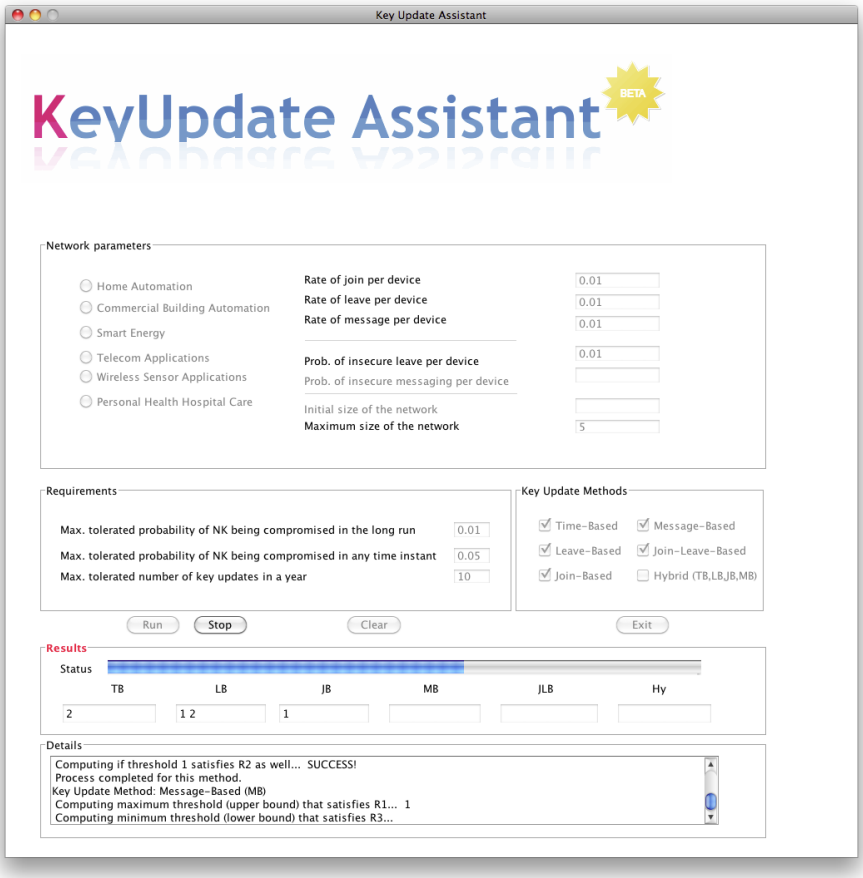


Figure 10.2: Key update assistant at work.

In Fig. 10.2, we present a screenshot of the tool running for a sample problem. We will demonstrate the tool with the help of this figure. The interface has five separate panels, three of them are for input and the rest is about the results. In the first panel, *Network parameters*, the user enters network information by either choosing an application profile (which makes things easy for ZigBee networks) or manually specifying the information as it is the case in Fig. 10.2. The next panel, *Requirements*, is for specifying the requirements to be verified. As we have explained in the design, the requirements have three different types regarding security and performance. The third panel, *Key Update Methods*,

enables the user to select the key update methods to be involved in the verification. For the sake of simplicity, only the methods mentioned and proposed in this dissertation are implemented.

After filling the first three panels, the user runs the assistant and the verification process starts. In the fourth panel, *Results*, the user can see the progress of the tool and the numerical values of satisfying thresholds for the methods being verified in corresponding boxes. For instance, in Fig. 10.2 TB is already verified and only threshold value of 2 is a solution for TB in the given case. The last panel, *Details*, is giving more information abut the process. For each selected key update method, maximum and minimum threshold values, satisfying threshold values, and any error is shown in this panel. For example, in Fig. 10.2 the tool has finished TB and continuing with LB and at the moment of screenshot the maximum threshold for LB is computed as 2 for the given case.

10.2 Protocol Verifier

In this section, we present our automated tool for analyzing certain qualitative security properties of protocols. Similar to the key update assistant, this tool also makes use of current techniques – in this case static program analysis – and makes formal verification to be useful for users with very limited or no expertise at all.

In Section 10.2.1 we explain the purpose of this tool, in Section 10.2.2 we present the details in the design of the tool, in Section 10.2.3 we detail the implementation phase, and finally in Section 10.2.4 we demonstrate the tool usage.

10.2.1 Purpose

Having a clear goal of verifying qualitative security properties and powerful technique of static program analysis, we pointed out distinct gaps in making use of formal verification in real protocols. Below we identify our purpose in developing a protocol verifier tool.

- It is not trivial to generate LySA processes out of protocol narrations. This task is normally done by hands and it is very much error–prone, and even worse any mistake in the LySA model cannot easily be spotted.

- A security protocol can be implemented in various settings. For example, implementing the bare protocol narration, implementing protocol narration after applying encryption or integrity or both to all messages, etc. This process should be automated to prevent modelling errors, and allow batch processing.
- Protocol narrations in so-called *Alice-Bob* notation still have ambiguities unless they are converted to extended protocol narrations as we have explained in Chaper 3. However, this conversion is also error-prone. Thus we need a more precise protocol narration format.

10.2.2 Design

In this section, we detail the design of the Protocol Verifier tool. In Section 10.2.2.1 we explain our protocol narration format, In Section 10.2.2.2 we explain the conversion from protocol narration to LySA process is achieved, and in Section 10.2.2.3 we explain the specific needs of different implementation policies depending on confidentiality and integrity requirements.

10.2.2.1 Protocol Narration Format

We start by extending the Alice–Bob notation to comply with our goals. Our first concern is in specifying elements of a message as *to be matched* and *to be bound*. We establish this by using semicolon instead of comma, similar to the LySA process calculus syntax. Thus, each message should have a semicolon to identify the two parts of the message.

Our second concern is in specifying pattern matching and variable binding with an encrypted component. An encrypted component may consist of at least one message elements. We establish this again by using semicolon in the proper position. Thus, each encrypted element should have a semicolon.

Even though there is no element to be matched, nor to be bound, there should be a semicolon inside each encrypted element and inside each message. We list usage examples of our new format below:

- **message with no elements to be matched:**
  1. A → B: `; element1`
- **message with no elements to be bound:**
  2. A → B: `element1;`

- **message with one element to be matched and one element to be bound:**
  3. A → B: `element1 ; element2`

- **message with two encrypted components:**
  4. A → B: `element1 ; element2,` $\{$`element3 ; element4`$\}_{key1}$`, { ; element5`$\}_{key2}$

10.2.2.2 LySa Process Generation

As we have mentioned before, a protocol narration should be extended to remove ambiguities so that it can be specified as a LySa process. We automate LySa process generation by focusing on the information of principals accumulated with each message, and optional annotations for each encryption decryption. These two points are the main points that can cause flaws in the model which will probably not be detected as long as the comply with LySa syntax. In other words, as this generation was done by hand, a smallest typo could cause a flawed protocol to pass the verification, or a secure protocol to fail the verification.

Below we list the issues that we focus in this phase of the design:

- A LySa model can be configured to have four different settings depending on the usage of identities and keys.

- A LySa model can be configured to have n number of groups to model different scenarios. Making $n = 3$ to be sufficient for a strong verification that covers man-in-the-middle attacks as well.

- User should be able to add any knowledge to any of the principals.

- A message should comply with the protocol narration format that we set in the previous section.

- The program should be able to identify a message element that is in the knowledge base of the principals and should replace that element with the corresponding variable if it is in the knowledge base.

- Using the knowledge base, program should detect which encrypted component can be decrypted by the receiver, and should issue proper decrypt clauses.

- Any encrypted component should come after the semicolon of a message.

We would like to note that, even though current tools (e.g. LyTe [O'S09]) can derive Alice-Bob notation from a given LySA process, deriving LySA process from a protocol narration is novel. Besides it is challenging in many cases such as Otway-Rees [OR87] protocol since received but not decrypted (due to lack of proper key) message components need to be saved and forwarded as intended in the design of the protocol.

10.2.2.3 Confidentiality and Integrity Implementations

Contemporary communication standards such as IEEE 802.15.4 allow different implementations of a security protocol, depending on security requirements. For example, a protocol having n messages can be implemented as it is specified, or by encrypting all the messages separately, or adding a message authentication code to the end of each message.

Based on our experience in ZigBee and IEEE 802.15.4., we decided to include four different protocol implementation policies. Doing so, a user will be able to verify different implementations of the same protocol with respect to confidentiality and integrity requirements.

- **Bare protocol:** The protocol narration is analyzed without any change.
- **Encryption:** Each message of the protocol is encrypted with the specified key.
- **Integrity:** For each message of the protocol, a message authentication key is computed and added to the message.
- **Encryption and Integrity:** Each message of the protocol is secured by both encryption and integrity.

As you may have noticed, the implementation choices above are sufficient to model all the protocol implementations in ZigBee and IEEE 802.15.4.

10.2.3 Implementation

We have implemented the protocol verifier as a tool that allows its users to specify a protocol easily and eliminating any user based errors in the resulting LySA model. The tool has a graphical user interface that a user can specify and add messages, configure both verification details (e.g. LySA settings on

identities, keys, etc.) and network details (e.g. confidentiality and integrity implementations, knowledge base of each principal etc.).

The tool uses LySA engine to apply static program analysis and pretty prints LySA verification results. The user will not be worried about the LySA model, and will have the opportunity to quickly revise the protocol itself or other configuration parameters. However, advanced users will also have the chance to modify the generated LySA models. Besides, the user can quickly establish a series of verifications as a batch process.

10.2.4 Demonstration

In this section, we present a demonstration of the tool we implemented. In addition to the design details above, we will present implementation details. We implemented the Protocol Verifier in Java, and tested in Mac OS X platform. The tool has a graphical user interface that allows its user to easily enter the input and see the results. In the background, the tool runs static program analysis which is implemented in LySA-tool in this case. The user does not need to know about how the program analysis engine is called, or how the LySA model is constructed. However, necessary information is produced and presented in the interface and besides all the technical details can be found in the log produced by non-verbose mode of the tool. All the implementation actually complies with our developments in the previous chapters.

Translating classical protocols such as Wide Mouthed Frog (WMF) in [BAN90] or Needham-Schroeder protocol [NS78] is fairly straightforward however many security protocols are more complicated to model in process algebra. As we have mentioned earlier, Otway-Rees is a difficult example that requires more effort to precisely model in LySA therefore we will perform our tool demonstration using this protocol. In Fig. 10.3, we present a screenshot of protocol verifier done with Otway-Rees protocol.

As seen in the figure, the graphical user interface of protocol verifier has four panels. Starting with the small panels at the left hand side, the *Principals* panel is where the user sets which principals (and attacker) are involved and similarly the *Configuration* panel is where the user configures LySA-tool to do verification for specified cluster numbers and encapsulation method.

The main and actually the largest panel is the *Messages* panel where the user is expected to enter the protocol to be verified as a narration that we described above. Each message of the protocol is entered one by one, and any message can be deleted easily. In Fig. 10.3, you can see that we have embedded some

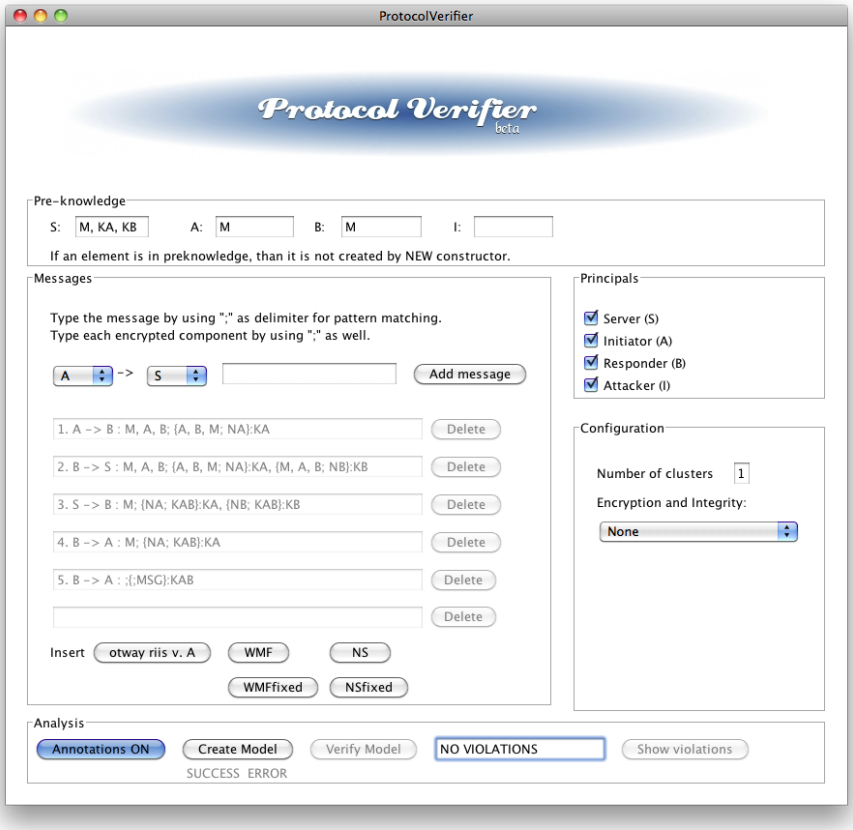


Figure 10.3: Protocol Verifier running Otway-Rees protocol.

well-known protocols for demonstration purposes. In the figure a specific version of Otway-Rees protocol is entered in our notation.

The *Pre-knowledge* panel is the next step in describing a protocol, although it is optional to add pre-knowledge. Any message element which is not defined as pre-knowledge is created by the *new* construct of LySA, in practice making it an unknown information for the attacker. For example, in our demonstration in Fig. 10.3 every legitimate principle knows M and server knows the keys of the principals.

After entering the protocol information to the mentioned panels in the user interface, we can proceed to the *Analysis* panel. Here the first thing before

creating a model is to set annotations switch, which is *ON* by default. Then by pushing the *Create Model* button the tool starts constructing LySA model for the given protocol with given verification configurations. If the messages were correctly entered and configuration was properly set the model will be created and *VERIFY MODEL* button will be available which is accompanied by a SUCCESS sign right below the Create Model button.

The last stage is the analysis, which is started by the *Verify Model* button. This is the moment where LySA-tool is called and static program analysis comes into play. As a result, the protocol is verified successfully if *NO VIOLATIONS* text appears. In case of possible violations detected, the user can see the details by pressing the *Show violations* button which will return LySA violation reporting.

Chapter 11

Conclusion

We shall have to evolve
problem-solvers galore —
since each problem they solve
creates ten problems more.

The Only Solution, Piet Hein

In this dissertation we have investigated the use of verification techniques powered by formal methods.

Our main thesis was:

> *Starting with a lower level of abstraction and applying a qualitative analysis on protocols, then integrating these formally verified protocols as components of scenarios in a higher level of abstraction followed by a quantitative analysis, we can achieve an efficient analysis scheme on information and communication systems.*

Our goal was to demonstrate that current techniques in formal verification can be combined in a smart way to be useful in the verification of communication

technologies, especially where limited–resources come in play and make things harder. In this dissertation we have taken a step towards achieving this goal, starting from qualitative analysis in order to verify discrete properties of the low level protocols and continuing with the quantitative analysis in order to verify quantitative aspects of the security-related scenarios where achieving absolute security is just impossible due to severe resource constraints and other factors depending on the type of the communication technologies.

In Section 11.1, based on our developments throughout this thesis we discuss our idea on building a framework for verification of communication standards. In Section 11.2, we summarize the contributions in this dissertation by relating them to our publications along the way. We conclude with final remarks followed by directions to future work in Section 11.3.

11.1 Towards A Framework For Verification of Communication Standards

In this section, we would like to discuss our idea on building a framework for verification of communication standards.

Assumption. We have a design for communication of devices, documented as a specification. A solid example can be given as network standards that define various layers of communication.

Problem. We want to be able to verify the design before actually implementing or running it. We have a property to be verified, which is often a security property such as secrecy, authenticity, or integrity.

Status. We are able to perform verification in two different abstraction levels. In High Level abstraction, we analyse the protocols using qualitative analysis and we verify that the protocols are confidentially secure or not. In Low Level abstraction, we analyse the scenarios using qualitative analysis and we verify the scenarios preserving secrecy with a probability satisfying our requirements.

In this dissertation, we have decided to set the abstraction levels as:

- *Verification in Low Level abstraction:* checking each protocol using static program analysis.
- *Verification in High Level Abstraction:* checking each scenario — composed of the protocols verified in the low level — using stochastic model

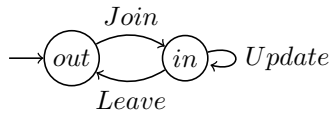
checking.

Below we present an example of our approach in verifying communication standards.

Notation. In the low level of abstraction we have protocols, which are translated into **actions** of **scenarios** in the high level abstraction. We denote an action as $\mathcal{A}ct\{\texttt{action\_name}\}$ and a scenario as $\mathcal{S}cen\{scenario\_name\}$. A scenarios is a concurrent system of modules synchronizing over certain actions.

We will denote the property to be verified as $\mathcal{P}rop\{property\_name\}$ In this example we will focus on secrecy property. An action that is verified in secrecy will be denoted as $\mathcal{A}ct\{\texttt{action\_name}\} \models_{spa} \mathcal{P}rop\{secrecy\}$. Notice that $_{spa}$ stands for the static program analysis, and will be replaced by $_{smc}$ in the case of scenarios where stochastic model checking is employed.

Example 11.1 (Secrecy in ZigBee Wireless Sensor networks) *In this example, we will first remind the state machine that we derived for the ZigBee security procedures in Section 2.2. In order to keep the example simple, we will cover a simple subset of it as shown below:*



This is a state machine for a single ZigBee device. When a device is out of the network, its state is out. A successful run of join protocol (with the trust center) will change the device's state as in. In this state, device can either update its key or leave the network (or may be removed). We focus on the security protocols which are shown as the transitions, and apply static program analysis to them. The results are:

- $\mathcal{A}ct\{\texttt{Join}\} \models_{spa} \mathcal{P}rop\{secrecy\}$
- $\mathcal{A}ct\{\texttt{Leave}\} \models_{spa} \mathcal{P}rop\{secrecy\}$
- $\mathcal{A}ct\{\texttt{Update}\}$[1] $\models_{spa} \mathcal{P}rop\{secrecy\}$

[1]Update protocol (in terms of rekeying, not recovery) does not exist in ZigBee, as we have identified in Chapter 3 and explained the details in Appendix B. However, we assume that an update protocol is developed, for example one of the six key update methods that we described in this thesis.

Then we proceed to a scenario where we have two modules: Network and UpdateStrategy. Network module is a type of birth-death process extended with the key update. UpdateStrategy is one of the key update methods we have proposed in this dissertations (i.e. LB, JB, JLB, Hy).

- $\mathcal{S}cen\{Mod\{Network\} \,||\, \mathcal{A}ct\{\texttt{Join}, \texttt{Leave}, \texttt{Update}\} \,||\, Mod\{UpdateStrategy\}\}$ $\models_{smc} \mathcal{P}rop\{secrecy\}$

Since we have formally verified the scenario which is composed of formally verified protocols, we can conclude that secrecy property is preserved if this scenario is employed. ■

In Fig. 11.1, we sketched the flow of our approach. We labelled the boxes to depict the stages, and below we will bind them to the relevant parts of this thesis:

- **ExtProc:** At this stage, we extract procedures out of a specification. In Chapter 2, we showed how to do this for ZigBee security sublayer.
- **FormProt:** At this stage, we formalize the protocols that will be used as building bricks for the scenarios. We presented a process algebraic way of doing this in Chapter 3.
- **LLAVer:** This is the stage where we apply Verification in Low Level Abstraction. In Chapter 3, we showed static program analysis as the verification method. We require qualitative analysis and 100% verification to pass this stage, and if we spot any flaw we go back to **FormProt** stage by correcting the protocols in stage **FixProt**.
  - **FixProt:** At this stage, we have to correct the flawed protocols. Since it is not a simple task and there is no solid method but good practices for fixing the protocols, we followed the guidelines from [AN94] and presented a real application in Chapter 3.
- **ConSce:** At this stage, we construct scenarios using fully verified protocols and formalize them allowing quantitative security analysis. We presented how to formalize a scenario in Markov chains in Chapter 5.
- **HLAVer:** This is the stage where we apply quantitative analysis to the scenarios we constructed. Instead of requiring strong verification, we expect quantitative guarantees as a result of this stage. We have given detailed examples for such qualitative analyses in Chapter 6. If the required guarantee levels are not satisfied then we have to correct or reconstruct the problematic scenarios in stage **FixSce**.

– **FixSce:** At this stage, we have to correct the flawed scenarios. Again it is not a simple task and there is no solid method, yet we gave examples on key update scenarios in Chapter 7 and Chapter 8 such that in case one key update method does not satisfy the requirements correction should be made as changing the key update method.

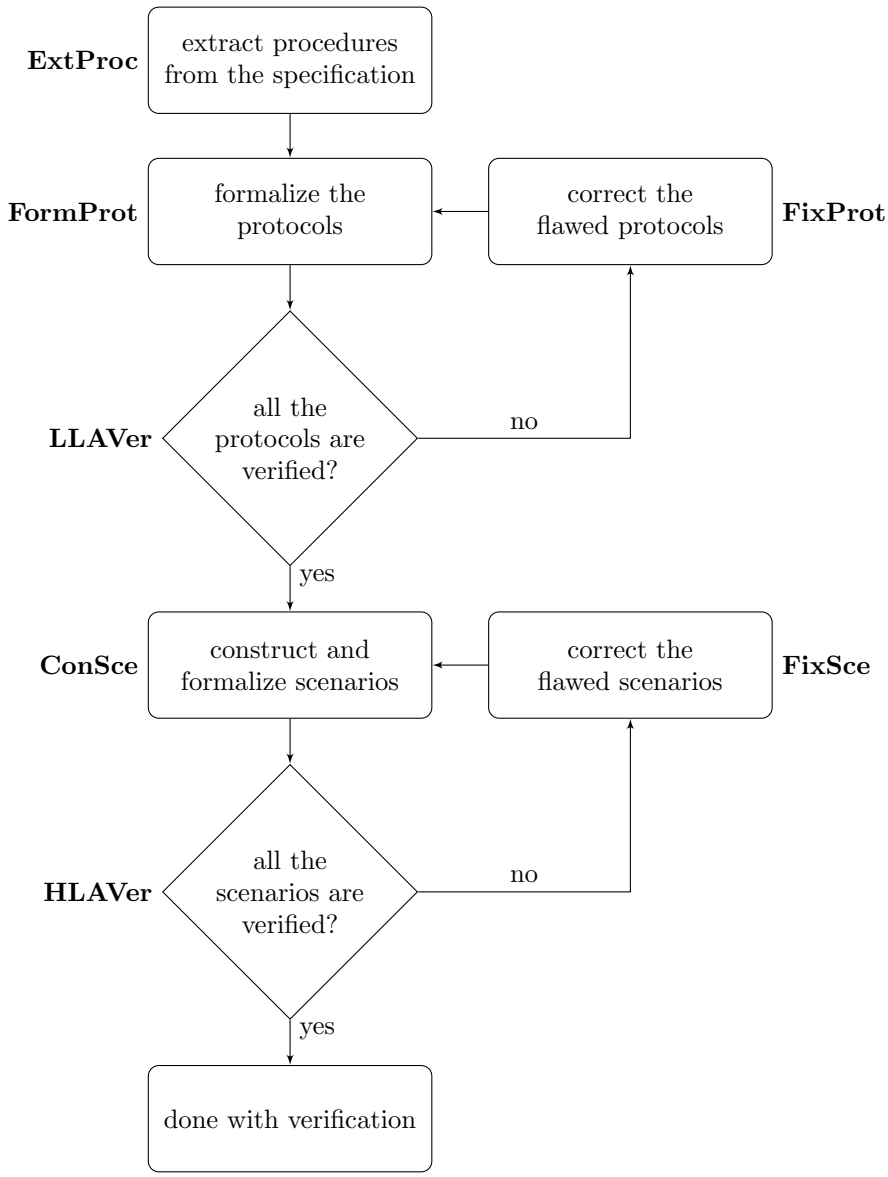


Figure 11.1: Verification in different abstraction levels

11.2 Contributions

In this section, we will recapitulate the contributions that we presented in this dissertations.

- [YNN08, YNN09b, YNN10a] We presented how to employ quantitative analysis techniques in verification of discrete security properties like confidentiality, integrity, and authenticity by picking a static program analysis approach. Along the way we contributed in a better understanding of the security layer of ZigBee, discovered a flaw in one of its critical protocols, and proposed a correction that is formally verified.
  - We investigated a wireless sensor network standard, ZigBee, from security perspective. Proceeding from specification to security procedures, and finally to security protocols, we clarified the security sublayer and derived protocol narrations for qualitative security analyses. We identified the gap in the ZigBee specification on key updates.
  - We presented the usage of static program analysis techniques in security protocol verification. We discovered a critical security flaw in a ZigBee security protocol which is currently in use. We fixed the protocol, and verified the fix also with static program analysis.
- [YNN09a, YNN$^+$10b] We proposed using quantitative analysis techniques in the verification of security protocols where trade–off between performance and security is inevitable. Along the way we contributed in key update area by developing methods, metrics, algorithms and analyses that are empowered by stochastic model checking.
  - We investigated the key update problem in ZigBee which is getting more difficult when the performance requirements are involved. We developed compact and scalable models that allow realistic quantitative analyses.
  - We developed metrics and methodologies for determining the optimum key update method and parameters.
  - We proposed new key update methods to be used with not only ZigBee but also in other resource-critical networking standards.
  - We analyzed various key update methods quantitatively and under different conditions. We proposed mechanisms in obtaining more efficient key updating schemes.
- [YNN11a, YNN11b, YNN11c] We presented case studies that realize our methodologies for determining optimum key update strategy for custom applications.

  - We demonstrated how to derive advice from quantitative analysis results in three different case studies focusing on three different application domains of wireless sensor networks.
  - We constructed an analysis to observe how different key update methods adapt to new environmental conditions in a network. We evaluated the methods and presented the results as a set of guidelines for network designers.
  - We proposed an adaptive mechanism to make a more efficient use of key update methods, considering power consumption.

- [Yük10] We designed and developed tools for improving security analyses and making them available to a wider range of users.
  - We designed and implemented a toolkit for LB key update method that computes the maximum risk in transient security analysis, and applies a transient analysis which covers only the necessary time period. The tool can be generalized for other key update methods, as well.
  - We designed and implemented a a software that automates the decision of key update method and parameters for a specific network. We created a push-button technology – powered by stochastic model checking – that network designers and even consumers can easily benefit from. Key Update Assistant, as we named it, runs necessary model checking operations and determines the optimum key update strategy that satisfies the requirements of its users.
  - We designed and implemented a protocol verifier that bridges the gap between protocol narrations and LySa models, and also automates qualitative verification using LySa engine. This tool allows verification of a protocol in various encapsulation forms involving integrity and encryption as in IEEE 802.15.4 and ZigBee.

11.3 Concluding Remarks And Future Work

In this thesis, we focused on the verification of the communication standards from a point of view that is close to the implementation and realisation. We used a real communication standard to demonstrate our developments. We stated a new approach that first verifies the low level protocols in a qualitative manner and guarantees absolute security, and then takes these verified protocols as actions of scenarios to be verified in a quantitative manner. Based on the emerging cyber–physical systems and resource–critical systems such as wireless

sensor networks, we used probabilistic verification that can return probabilistic results with respect to the trade–off between security and performance.

We had a vision in the beginning of the thesis: we should bridge the gap between the theoretical developments and the practical implementations. We believe that we succeeded in this way, by showing how to employ the state–of–the–art verification techniques in specific problems related to communication technologies. In this sense, we have extended various existing ideas and also proposed new ideas to improve verification. Especially in the problem of key update, we believe we have contributed to the solution for not only wireless sensor networks but also many other types of systems that require key updates. Besides we produced automated tools that were intended to demonstrate what kind of tools can developed on different purposes and application domains, however resulting tools can already be used in many types of networks. For example our tool key update assistant provides a push–button technology that can be used by people that has very limited knowledge or no knowledge at all in verification technology.

At this last section of the dissertation we would like to discuss directions for the future work. Since verification is a broad topic, many aspects remain to be addressed even if we limit it to the verification of security protocols. Based on our discussion on a framework including two layers of abstraction, we consider future work to take place in this direction. In the qualitative analysis of security protocols, two powerful methods program analysis and model checking can be combined in a way to benefit from the advantages of both of the methods. Thus, the state space explosion problem in model checking and lack of trace problem in program analysis can be eliminated. In the quantitative analysis, beyond all theoretical problems in modelling and computation, we face a complication in supplying the realistic input to the tools (e.g. reliability value per device, messaging frequency per device, etc.). Developing an approach in determining the proper set of input data, and eventually getting a set of results is one of the points that needs attention.

Appendix A

Protocol Narrations Derived From ZigBee Security Sublayer

A.1 Protocol Narrations

A.1.1 Guide For Reading Narrations

Protocol Narration Convention Extending the classical *Alice-Bob* protocol narration style, we used message (primitive) titles in boldface fonts, and message fields inside angle brackets. The protocol narration convention is given below:

	Format: A. B→C: D-E.F <M>
A:	Message Number (*1,2,3,...*)
B and C:	Sender and Receiver (Trust Center: *TC*, Router: *R*, All Routers: *Rn*, Device: *D*, All Devices: *Dn*, Initiator: *I*, Responder: *Res*)
D:	Message Layer and Entity (APS Management Entity: *APSME*, NWK Management Entity: *NLME*)
E:	Command Type (*SWITCH-KEY, ESTABLISH-KEY, etc.*)
F:	Primitive Type (*request, response*)
M:	Message structure, including field names (*Accept, SrcAddress, etc.*) and possible values (*True, Raddr, etc.*)

Example:

3. D→TC: APSME-ESTABLISH-KEY.response

<InitiatorAddress(=TCaddr), Accept(=TRUE)>

This is the third message of the protocol, sent by a device to the TC, the message is an APS layer management entity message, the command type is *establish key*, the primitive type is *response*. The message has two fields, the first field *InitiatorAddress* has the value *TCaddr*, and the second field *Accept* has the value *TRUE*.

Cryptographic Notation We used three different cryptographic operations: *Encryption*, *HMAC*, and *Hash*. The notation is given below:

{Message}:Key : *Message* is encrypted with *Key*
MAC{Message}:MacKey : HMAC of *Message*, computed using *MacKey*
H(Message) : Hash of *Message*

A.1.2 Join

The Join procedure depends on the MAC layer command frames that are defined in the IEEE 802.15.4 standard, and being the only exception in this appendix, we omitted the message details.

1. D → Rn: Beacon request command (unsecured)

2. Rn → D: Beacon (unsecured)

3. D → R: Association request command

4. R → D: Association response command

A.1.3 Authentication

Protocol Naming Convention In order to have a shorter and a systematic way of naming protocols, we created the protocol naming convention below for the Authentication protocols:

Format: G-H-I[-J]
G: Security Mode (Standard Security: *SS*, High Security: *HS*)
H: Presence of a separate router
(A separate router exists between TC and D: *R*,
TC serves as the router: *NR*)
I: Key knowledge of the device
(Preconfigured with a key: *PK*, Not preconfigured: *NPK*)
J: Preconfigured or "to be configured" key type
(None, Network Key: *NK*, Trust Center Link Key: *TCLK*,
Trust Center Master Key: *TCMK*)

Example: SS-R-PK-TCLK means:
Standard Security Mode,
TC is not the router,
Device is preconfigured with TCLK

The protocols in the Authentication part can be summarized as in Fig. A.1. In order to save space, we present the narrations in the case that TC is the router.

SS-NR-NPK

1. TC→D: APSME-TRANSPORT-KEY.request (send active SNK)

<DestAddress(=Daddr), Keytype(=SNKtype), KeySeqNumber(=activeKeySeqNo), NetworkKey(=activeSNK),

UseParent(=FALSE), ParentAddress(=No need to set)>

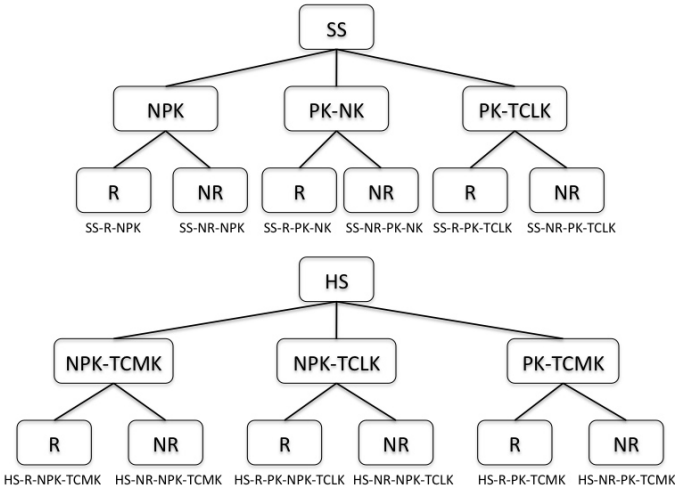


Figure A.1: Authentication Protocols

SS-NR-PK-NK

1. TC→D: APSME-TRANSPORT-KEY.request (send active SNK)

<DestAddress(=Daddr), Keytype(=SNKtype), KeySeqNumber(=Zero), NetworkKey(=Zero), UseParent(=FALSE),

ParentAddress(=No need to set)>

SS-NR-PK-TCLK

1. TC→D: APSME-TRANSPORT-KEY.request (send active SNK)

<{DestAddress(=Daddr), Keytype(=SNKtype), KeySeqNumber(=activeKeySeqNo), NetworkKey(=activeSNK),

UseParent(=FALSE), ParentAddress(=No need to set)}:TCLK>

HS-NR-NPK-TCMK

1. TC→D: APSME-TRANSPORT-KEY.request (send TCMK)

<DestAddress(=Daddr), Keytype(=TCMKtype), ParentAddress(=TCaddr), Key(=TCMK)>

2. TC→D: APSME-ESTABLISH-KEY.request (establish TCLK)

<ResponderAddress(=Daddr), UseParent(=FALSE), ResponderParentAddress(=No need to set),

KeyEstablishmentMethod(=SKKE)>

3. D→TC: APSME-ESTABLISH-KEY.response (establish TCLK)

<InitiatorAddress(=TCaddr), Accept(=TRUE)>

4. TC↔D: SKKE (*See Subsection 2.1.5.2 for details*)

5. TC→D: APSME-TRANSPORT-KEY.request (send active HNK)

<DestAddress(=Daddr), Keytype(=HNKtype), KeySeqNumber(=activeKeySeqNo), NetworkKey(=activeHNK), UseParent(=FALSE), ParentAddress(=No need to set)>

6. D→TC: APSME-AUTHENTICATION.request (entity auth.)

<PartnerAddress(=TCaddr), Action(=INITIATE), RandomChallenge(=Drandom)>

7. TC→D: APSME-AUTHENTICATION.request (entity auth.)

<PartnerAddress(=Daddr), Action(=RESPOND_ACCEPT), RandomChallenge(=TCrandom)>

8. D↔TC: MEA (*See Subsection 2.1.5.3 for details*)

HS-NR-NPK-TCLK

1. TC→D: APSME-TRANSPORT-KEY.request (send TCLK)

<DestAddress(=Daddr), Keytype(=TCLKtype), ParentAddress(=TCaddr), Key(=TCLK)>

2. TC→D: APSME-TRANSPORT-KEY.request (send active HNK)

<DestAddress(=Daddr), Keytype(=HNKtype), KeySeqNumber(=activeKeySeqNo), NetworkKey(=activeHNK), UseParent(=FALSE), ParentAddress(=No need to set)>

3. D→TC: APSME-AUTHENTICATION.request (entity auth.)

<PartnerAddress(=TCaddr), Action(=INITIATE), RandomChallenge(=Drandom)>

4. TC→D: APSME-AUTHENTICATION.request (entity auth.)

<PartnerAddress(=Daddr), Action(=RESPOND_ACCEPT), RandomChallenge(=TCrandom)>

5. D↔TC: MEA (*See Subsection 2.1.5.3 for details*)

HS-NR-PK-TCMK

1. TC→D: APSME-ESTABLISH-KEY.request (establish TCLK)

<ResponderAddress(=Daddr), UseParent(=FALSE), ResponderParentAddress(=No need to set), KeyEstablishmentMethod(=SKKE)>

2. D→TC: APSME-ESTABLISH-KEY.response (establish TCLK)

<InitiatorAddress(=TCaddr), Accept(=TRUE)>

3. TC↔D: SKKE (*See Subsection 2.1.5.2 for details*)

4. TC→D: APSME-TRANSPORT-KEY.request (send active HNK)

<DestAddress(=Daddr), Keytype(=HNKtype), KeySeqNumber(=activeKeySeqNo), NetworkKey(=activeHNK),

UseParent(=FALSE), ParentAddress(=No need to set)>

5. D→TC: APSME-AUTHENTICATION.request (entity auth.)

<PartnerAddress(=TCaddr), Action(=INITIATE), RandomChallenge(=Drandom)>

6. TC→D: APSME-AUTHENTICATION.request (entity auth.)

<PartnerAddress(=Daddr), Action(=RESPOND_ACCEPT), RandomChallenge(=TCrandom)>

7. D↔TC: MEA (*See Subsection 2.1.5.3 for details*)

A.1.4 NK Update

We have two different NK Update protocols for SS and HS modes.

SS

1. TC→Dn: APSME-TRANSPORT-KEY.request

<DestAddress(=Broadcastaddr), KeyType(=SNKtype), KeySeqNumber(=newKeySeqNumber), NetworkKey(=newSNK),

UseParent(=FALSE), ParentAddress(=No need to set)>

2. TC→Dn: APSME-SWITCH-KEY.request

<DestAddress(=Broadcastaddr), KeySeqNumber(=newKeySeqNumber)>

HS

1. TC→D: APSME-TRANSPORT-KEY.request

<DestAddress(=Daddr), KeyType(=HSNKtype), KeySeqNumber(=newKeySeqNumber), NetworkKey(=newHSNK),

UseParent(=FALSE), ParentAddress(=No need to set)>

2. TC→D: APSME-SWITCH-KEY.request

<DestAddress(=Daddr), KeySeqNumber(=newKeySeqNumber)>

A.1.5 End-to-End Application Key Establishment

This procedure depends on the configuration of the TC. Therefore, we have two different protocols.

TC is configured to send AppLK

1. I→TC: APSME-REQUEST-KEY.request

<DestAddress(=Resaddr), KeyType(=AppKeytype), PartnerAddress(=Resaddr)>

2. TC→I: APSME-TRANSPORT-KEY.request

<DestAddress(=Iaddr), KeyType(=AppLKtype), PartnerAddress(=Resaddr), Initiator(=TRUE), Key(=newAppLK)>

3. TC→Res: APSME-TRANSPORT-KEY.request

<DestAddress(=Resaddr), KeyType(=AppLKtype), PartnerAddress(=Iaddr), Initiator(=FALSE), Key(=newAppLK)>

TC is configured to send AppMK

1. I→TC: APSME-REQUEST-KEY.request

<DestAddress(=Resaddr), KeyType(=AppKeytype), PartnerAddress(=Resaddr)>

2. TC→I: APSME-TRANSPORT-KEY.request

<DestAddress(=Iaddr), KeyType(=AppMKtype), PartnerAddress(=Resaddr), Initiator(=TRUE), Key(=newAppMK)>

3. TC→Res: APSME-TRANSPORT-KEY.request

<DestAddress(=Resaddr), KeyType(=AppMKtype), PartnerAddress(=Iaddr), Initiator(=FALSE), Key(=newAppMK)>

4. I→Res: APSME-ESTABLISH-KEY.request (establish TCLK)

<ResponderAddress(=Resaddr), UseParent(=FALSE), ResponderParentAddress(=No need to set),

KeyEstablishmentMethod(=SKKE)>

5. Res→I: APSME-ESTABLISH-KEY.response (establish TCLK)

<InitiatorAddress(=Iaddr), Accept(=TRUE)>

6. I↔Res: SKKE

A.1.6 Network Leave

This procedure has different protocols, depending on the initiator of the procedure.

Remove-Device

1. TC→R: APSME-REMOVE-DEVICE.request

<ParentAddress(=Raddr), ChildAddress(=Daddr)>

2. R→D: NLME-LEAVE.request

<DeviceAddress(=Daddr), RemoveChildren(=TRUE/FALSE), Rejoin(=FALSE)>

Device-Leave

1. D→R: NLME-LEAVE.request

<DeviceAddress(=Daddr), RemoveChildren(=TRUE/FALSE), Rejoin(=FALSE)>

2. R→TC: APSME-UPDATE-DEVICE.request

<SrcAddress(=Raddr), DeviceAddress(=Daddr), Status(=DeviceLeft), DeviceShortAddress(=Dshortaddr)>

Appendix B

Key Update in ZigBee

B.1 The Gap in the ZigBee Specification

The latest ZigBee Specification (**ZigBee-2007**) [Zig08d], states that "*policy decisions to **expire and periodically update keys, if desired** must be addressed correctly by any real implementation*". In addition, it is stated that "*the **application profiles**[1] should include these policies*"[2].

The latest published ZigBee application profile is the *Smart Energy Application Profile* (**ZigBee-SE**) [Zig08c], which is curently the most critical and important ZigBee application profile. ZigBee-SE states that "*Periodically the trust center **shall** update the **network key (NK)***" and "*Periodically the trust center **may** update the **link key** associated with a particular device.*"[3].

In addition to the specification documents (i.e. ZigBee-2007) and application profiles (e.g. ZigBee-SE), one should also consider the *Stack Profile*s. Stack

[1] An *Application Profile* is defined as "*a collection of device descriptions, which together form a cooperative application.*" in the ZigBee Specification. Application profiles provide standard interfaces and device definitions to allow interoperability among ZigBee devices produced by various manufacturers, in a specific application domain.

[2] [Zig08d], Section 4.2.1.2 Security Design Choices, page 422.

[3] [Zig08c], Section 5.4 Smart Energy Profile Security, page 20. As a note on conformance levels, *may* equals *is permitted*, and *shall* equals *is required to* in the ZigBee-SE Specification.

profiles are intended to support the *Application Profiles*, since most of the parameters that are defined in the application profiles are set by the stack profiles. The latest published ZigBee-PRO Stack Profile [Zig08b] states that "*it is **recommended** that the trust center change the network key if it is discovered that any device has been stolen or otherwise compromised, ...*" and "*there is **no expectation** that the network key be changed when **adding a new device**.*".

As seen clearly from our explanations with precise references above, all the specification documents of ZigBee leave the important key update issue to the implementations.

B.2 ZigBee Application Profiles

Currently, ZigBee has six different application profiles and up to now only two of them are finalized, *Home Automation* [Zig08a] and *Smart Energy* [Zig08c]. The remaining application profiles that are expected to be finalized and released soon are *Commercial Building Automation*, *Personal, Home and Hospital Care*, *Telecom Applications*, and *Wireless Sensor Applications*.

In this section, we explain the settings for the application profiles that we focused on in this thesis. For each application profile, we first summarize the scope and the purpose of the profile for a better understanding of the design criteria. Then in the second paragraph, we present the information gathered from a professional ZigBee expert that once led the design of ZigBee security sublayer [Cra08], and finally present the parameters that we used in our models with the values that we determined from that information. Specifically, we present the technical details in terms of maximum network size, rate of join, rate of leave, and key compromise probability.

The Home Automation (HA) Profile: This profile covers applications for the residential automation market to allow original equipment manufacturers (OEM) to produce products that will meet the needs of customers ranging from do-it-yourself homeowners to professional installers. Home automation profile has a vast amount of device types such as on/off switch, level control switch, remote control, shade controller, heating cooling unit, various sensors (temperature, pressure, light), intruder alarm system equipments etc.

In this profile, the network is *fairly static* such that it is likely that devices such as light switches and luminaries, once commissioned, would remain in place for a longer period. The network size is in general less than 50 devices. The

environment is relatively insecure, and to reflect this we shall say the key is compromised in 1% of the cases. A device may leave the network for reasons such as a break down or flat battery, and most likely, it will be replaced shortly after. We shall assume that each device leaves the network once a year but it will be replaced within a week. Based on these assumptions we specify the remaining constants as follows:

```
maximal size of the network: 20 devices (excluding the trust center)
average number of joining devices: 1 device every week
average number of leaving devices: 1 device per year
risk of key compromise: 1/100
```

The Smart Energy (SE) Profile: This profile covers applications for two-way communications of metering data and energy management to provide more efficient and reliable energy usage. Smart Energy profile goes beyond automated meter reading to demand-response systems for real-time pricing and voluntary load shedding. Example devices include but not limited to energy services portal, metering end device, in-premise display, etc.

In this profile, the network is *static*, so the devices rarely leave the network. The network size is in general 3-5 devices, including an Energy Service Portal in a meter, a Programmable Communicating Thermostat, and a display. The environment is highly secure, and to reflect this the risk of key compromise is very low. Once a device has left the network, it will be replaced shortly. Based on these assumptions we specify the remaining constants as follows:

```
maximal size of the network: 5 devices (excluding the trust center)
average number of joining devices: 1 device every week
average number of leaving devices: 1 device per five years
risk of key compromise: 1/10000
```

The Commercial Building Automation (CBA) Profile: This profile covers applications targeted at a commercial building environment such that: having a coverage area of up to 100,000 square feet or more, typically professionally managed, may have unrestricted access with attendant security implications, inter-working with an installed base of existing products on other networks, etc. Example devices include but not limited to ballast unit, commissioning tool, occupacy and light sensors, thermostat, etc.

Also in this profile, the network is *fairly static* but its size is somewhat higher than in the previous profiles although it is in general less than 200 devices. The environment is relatively secure, and also here left devices will be replaced

within a short period. Based on these assumptions we specify the remaining constants as follows:

```
maximal size of the network: 100 devices (excluding the trust center)
average number of joining devices: 1 device every week
average number of leaving devices: 1 device per year
risk of key compromise: 1/1000
```

The Personal, Home and Hospital Care (PHHC) Profile: This profile will be used by all the devices which jointly cooperate to fulfill the requirements of a non-invasive health care application. The devices involved in a health care application could be classified as medical devices (blood pressure monitor, oxygen saturation monitor, EEG, etc.) and non-medical devices (gateway, cell phone, light system, etc.). The health care application use cases can be logically separated into the following categories: Chronic disease monitoring, Personal wellness monitoring (ensuring an individual's wellness and safety), and Physical fitness.

In this profile, the network is *dynamic* such that it is likely that devices join and leave the network regularly. The network size is in general less than 1000 devices. The environment is very secure, still the key updates are needed to be rather frequent. Based on these assumptions we specify the remaining constants as follows:

```
maximal size of the network: 500 devices (excluding the trust center)
average number of joining devices: 1 device every week
average number of leaving devices: 1 device per month
risk of key compromise: 1/10000
```

The Telecom Applications (TA) Profile: This profile will be applied in telecom value-added services and supplementary services to enhance and fulfill the telecom network functions, and it also includes some applications integrated with some mobile terminals and plug-in modules. Example devices using this profile can be mobile terminals, SIM cards, and Point-of-Sale (PoS) devices.

In this profile, the network is *dynamic* such that the nodes are joining and leaving frequently. The network size is in general less than 50 devices. The environment is highly secure, therefore key updates are needed to be rather infrequent. Based on these assumptions we specify the remaining constants as follows:

```
maximal size of the network: 20 devices (excluding the trust center)
```

```
average number of joining devices: 1 device every week
average number of leaving devices: 1 device per month
risk of key compromise: 1/100000
```

The Wireless Sensor Applications (WSA) Profile: This profile is designed to enable wireless sensor network applications such as *environmental monitoring* of either indoor or outdoor areas, *asset tracking* of mobile tagged-things or persons, and *structural or machine monitoring*. Installation profiles may include areas with little or no infrastructure such as networks that are deployed outdoors, or temporary installations. In such situations, the assumption of a powered backbone is no longer tenable. The nodes themselves should be capable of battery powered routing or forwarding of their neighbors' data to one or more centralized collection points. Note that with sensor data, the key assumption is that the data must be collected and pulled out of the network. Example devices that could be based on this profile include acoustic/ultrasonic output device, binary output device, and relevant sensors to measure related parameters.

In this profile, the network is *fairly dynamic* though it depends on the particular application. The network size is in general less than 1000 devices. The environment is secure, still the key updates are needed to be rather frequent. Based on these assumptions we specify the remaining constants as follows:

```
maximal size of the network: 500 devices (excluding the trust center)
average number of joining devices: 1 device every week
average number of leaving devices: 1 device per 6 months
risk of key compromise: 1/1000
```

Obviously, application profiles can be customized easily and the models can also be used for different type of networks.

Appendix C

Key Update Models in PRISM

In this appendix, we will list the PRISM models that we implemented various key update strategies. We have used these models mostly in Chapters 5, 6, 7, 8 and also in Chapters 9, 10. Besides we will list the stochastic temporal logic (CSL, CSRL, etc.) formulae that we used for property specification.

C.1 Models Considering Key Compromise By Leaving Devices Only

C.1.1 Join-based Key Update - *model-L-join.sm*

```
ctmc

const int J; // threshold for number of joins

// time unit: 1 day
const double R_join   = 1/7;   // average join: 1 every week

// Custom values!
const int Max=200;
const double R_leave  = 1/7;
const double P_comp   = 1/1000;

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [joinR]  Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
endmodule

module COORDINATOR
 Comp: bool init false;
 C_join: [0..J] init 0;

 [join]   C_join<J-1 -> (C_join'=C_join+1);
 [joinR]  C_join=J-1 -> (C_join'=0) & (Comp'=false);
 [leave]  true -> true;
 [leaveC] true -> (Comp'=true);

endmodule
```

C.1.2 Join-Leave-based Key Update - *model-L-joinleave.sm*

```
ctmc

const int N; // threshold for number of leaves

// time unit: 1 day
const double R_join   = 1/7;   // average join: 1 every week

// values for ha
const int Max=20;
const double R_leave  = 1/365; // average leave: 1 per year
const double P_comp   = 1/100; // risk of key leakage

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [joinR]  Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [leaveR] Size>0    -> R_leave*Size:      (Size'=Size-1);
endmodule

module COORDINATOR
 Comp: bool init false;
 C_joinleave: [0..N] init 0;

 [join]   C_joinleave<N-1 -> (C_joinleave'=C_joinleave+1);
 [joinR]  C_joinleave=N-1 -> (C_joinleave'=C_joinleave+1);
 [leave]  C_joinleave<N-1 -> (C_joinleave'=C_joinleave+1);
 [leaveC] C_joinleave<N-1 -> (C_joinleave'=C_joinleave+1) & (Comp'=true);
 [leaveR] C_joinleave=N-1 -> (C_joinleave'=0) & (Comp'=false);
endmodule
```

C.1.3 Leave-based Key Update - *model-L-leave.sm*

```
ctmc

const int N; // threshold for number of leaves

// time unit: 1 day
const double R_join   = 1/7;   // average join: 1 every week


// Custom values
const int Max=200;
const double R_leave  = 1/7;
const double P_comp   = 1/1000;

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [leaveR] Size>0    -> R_leave*Size:      (Size'=Size-1);
endmodule

module COORDINATOR
 Comp: bool init false;
 C_leave: [0..N] init 0;

 [join]   true -> true;
 [leave]  C_leave<N-1 -> (C_leave'=C_leave+1);
 [leaveC] C_leave<N-1  -> (C_leave'=C_leave+1) & (Comp'=true);
 [leaveR] C_leave=N-1-> (C_leave'=0) & (Comp'=false);
endmodule
```

C.1.4 Message-based Key Update - *model-L-message.sm*

```
ctmc

const int MSG; // threshold for number of messages

// time unit: 1 day
const double R_join   = 1/7;   // average join: 1 every week

// Custom values
const int Max=200;
const double R_leave  = 1/7;
const double P_comp   = 1/1000;
const double R_message  = 1; // average communication per device

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [message] Size>0    -> R_message*Size:      true;
 [messageR] Size>0    -> R_message*Size:      true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_msg: [0..MSG] init 0;

 [join]   true -> true;
 [leave]  true -> true;
 [leaveC] true  -> (Comp'=true);
 [message] C_msg<MSG-1  -> (C_msg'=C_msg+1);
 [messageR] C_msg=MSG-1 -> (C_msg'=0) & (Comp'=false);

endmodule
```

C.1.5 Time-based Key Update - *model-L-time.sm*

```
ctmc

const int M; // number of months between resets

// time unit: 1 day
const double R_join   = 1/7;   // average join: 1 every week
const int k;
const double mean = 30*M;

//values for ha
const int Max=20;
const double R_leave  = 1/365; // average leave: 1 per year
const double P_comp   = 1/100; // risk of key leakage

module DEVICES
 Size: [0..Max] init Max;

 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [reset]  true      -> 1:                       true;
endmodule

module COORDINATOR
 Comp: bool init false;
 i : [1..k+1];

 [join]   true ->                true;
 [leave]  true ->                true;
 [leaveC] true ->                (Comp'=true);
 []   i < k -> k/mean :   (i'=i+1);
 [reset] i = k -> k/mean : (i'=1) & (Comp'=false);
endmodule
```

C.2 Models Considering Key Compromise By Leaving Devices and Sent Messages

C.2.1 Join-based Key Update - *model-LM-join.sm*

```
ctmc

const int N; // threshold for number of leaves

// time unit: 1 day
//const double R_join   = 1/7;   // average join: 1 every week

// TEST PARAMETERS
const int    Max=50;
const double R_leave  = 1/365; // average leave: 1 per year
const double R_join   = 1/2;   // average join: replacement in two days
const double R_message  = 1; // average communication: 4 times a day
const double P_comp   = 1/10000; // risk of key leakage

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [joinR]  Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_join: [0..N] init 0;

 [join]   C_join<N-1 -> (C_join'=C_join+1);
 [joinR]  C_join=N-1-> (C_join'=0) & (Comp'=false);
 [leave]  true -> true;
 [leaveC] true -> (Comp'=true);
 [message] true  ->              true;
 [messageC] true ->              (Comp'=true);
endmodule
```

C.2.2 Join-Leave-based Key Update - *model-LM-joinleave.sm*

```
ctmc

const int N; // threshold for number of leave and joins

// time unit: 1 day
//const double R_join   = 1/7;   // average join: 1 every week


// TEST VALUES
const int    Max=50;
const double R_leave  = 1/365; // average leave: 1 per year
const double R_join   = 1/2;   // average join: replacement in two days
const double R_message  = 1; // average communication: 4 times a day
const double P_comp   = 1/10000; // risk of key leakage

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [joinR]  Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [leaveR] Size>0    -> R_leave*Size:      (Size'=Size-1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_joinleave: [0..N] init 0;

 [join]   C_joinleave<N-1 -> (C_joinleave'=C_joinleave+1);
 [joinR]  C_joinleave=N-1 -> (C_joinleave'=0) & (Comp'=false);
 [leave]  C_joinleave<N-1 -> (C_joinleave'=C_joinleave+1);
 [leaveC] C_joinleave<N-1 -> (C_joinleave'=C_joinleave+1) & (Comp'=true);
 [leaveR] C_joinleave=N-1 -> (C_joinleave'=0) & (Comp'=false);
 [message] true  ->              true;
 [messageC] true ->              (Comp'=true);
endmodule
```

C.2.3 Leave-based Key Update - *model-LM-leave.sm*

```
ctmc

const int N; // threshold for number of leaves

// time unit: 1 day
//const double R_join   = 1/7;   // average join: 1 every week

// TEST PARAMETERS
const int    Max=50;
const double R_leave  = 1/365; // average leave: 1 per year
const double R_join   = 1/2;   // average join: replacement in two days
const double R_message  = 1; // average communication: 4 times a day
const double P_comp   = 1/10000; // risk of key leakage


module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [leaveR] Size>0    -> R_leave*Size:      (Size'=Size-1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_leave: [0..N] init 0;

 [join]   true -> true;
 [leave]  C_leave<N-1 -> (C_leave'=C_leave+1);
 [leaveC] C_leave<N-1  -> (C_leave'=C_leave+1) & (Comp'=true);
 [leaveR] C_leave=N-1-> (C_leave'=0) & (Comp'=false);
 [message] true  ->              true;
 [messageC] true ->              (Comp'=true);
endmodule
```

C.2.4 Message-based Key Update - *model-LM-msg.sm*

```
ctmc

const int MSG; // threshold for number of messages

// time unit: 1 day

// TEST PARAMETERS
const int    Max=50;
const double R_leave  = 1/365; // average leave: 1 per year
const double R_join   = 1/2;   // average join: replacement in two days
const double R_message  = 1; // average communication: 4 times a day
const double P_comp   = 1/10000; // risk of key leakage

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
 [messageR] Size>0    -> R_message*Size:      true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_msg: [0..MSG] init 0;

 [join]   true -> true;
 [leave]  true -> true;
 [leaveC] true  -> (Comp'=true);
 [message] C_msg<MSG-1  -> (C_msg'=C_msg+1);
 [messageC] C_msg<MSG-1 -> (C_msg'=C_msg+1) & (Comp'=true);
 [messageR] C_msg=MSG-1 -> (C_msg'=0) & (Comp'=false);

endmodule
```

C.2.5 Time-based Key Update - *model-LM-join.sm*

```
ctmc

const int M; // number of months between resets

// time unit: 1 day
const int k;
const double mean = 30*M;

// TEST PARAMETERS
const int    Max=50;
const double R_leave  = 1/365; // average leave: 1 per year
const double R_join   = 1/2;   // average join: replacement in two days
const double R_message  = 1; // average communication: 4 times a day
const double P_comp   = 1/10000; // risk of key leakage


module DEVICES
 Size: [0..Max] init Max;

 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
 [reset]  true      -> 1:                        true;
endmodule

module COORDINATOR
 Comp: bool init false;
 i : [1..k+1];

 [join]   true ->                true;
 [leave]  true ->                true;
 [leaveC] true ->                (Comp'=true);
 []   i < k -> k/mean :   (i'=i+1);
 [message] true  ->              true;
 [messageC] true ->              (Comp'=true);
 [reset] i = k -> k/mean : (i'=1) & (Comp'=false);
endmodule
```

C.2.6 Hybrid Key Update - *model-LM-hybrid.sm*

```
ctmc

const int J; // threshold for number of joins
const int N; // threshold for number of leaves
const int MSG; // threshold for number of messages
const int M; // number of months between resets
const int k;
const double mean = 30*M;

// TEST PARAMETERS
const int Max=50;
const double R_join   = 1/2;   // average join: 1 every week
const double R_leave  = 1/365; // average leave: 1 per year
const double P_comp   = 1/10000; // risk of key leakage
const double R_message  = 1; // average communication

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [joinR]  Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [leaveR] Size>0    -> R_leave*Size:      (Size'=Size-1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
 [messageR] Size>0    -> R_message*Size:      true;
 [reset]  true      -> 1:                       true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_join: [0..J] init 0;
 C_leave: [0..N] init 0;
 C_msg: [0..MSG] init 0;
 i : [1..k+1];

 [join]   C_join<J-1 -> (C_join'=C_join+1);
 [joinR]  C_join=J-1-> (C_join'=0)&(C_leave'=0)&(C_msg'=0)&(i'=1)&(Comp'=false);
 [leave]  C_leave<N-1 -> (C_leave'=C_leave+1);
 [leaveC] C_leave<N-1 -> (C_leave'=C_leave+1) & (Comp'=true);
 [leaveR] C_leave=N-1-> (C_join'=0)&(C_leave'=0)&(C_msg'=0)&(i'=1)&(Comp'=false);
 [message] C_msg<MSG-1  -> (C_msg'=C_msg+1);
 [messageC] C_msg<MSG-1 -> (C_msg'=C_msg+1) & (Comp'=true);
 [messageR] C_msg=MSG-1 -> (C_join'=0)&(C_leave'=0)&(C_msg'=0)&(i'=1)&(Comp'=false);
 []   i < k -> k/mean :   (i'=i+1);
 [reset] i=k -> k/mean:(C_join'=0)&(C_leave'=0)&(C_msg'=0)&(i'=1)&(Comp'=false);
endmodule
```

C.2.7 Hybrid Key Update excluding MB - *model-LM-hybrid—mb.sm*

```
ctmc

const int J; // threshold for number of joins
const int N; // threshold for number of leaves
const int M; // number of months between resets
const int k;
const double mean = 30*M;

// TEST PARAMETERS
const int Max=50;
const double R_join   = 1/2;   // average join: 1 every week
const double R_leave  = 1/365; // average leave: 1 per year
const double P_comp   = 1/10000; // risk of key leakage
const double R_message  = 1; // average communication

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [joinR]  Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [leaveR] Size>0    -> R_leave*Size:      (Size'=Size-1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
 [reset]  true      -> 1:                       true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_join: [0..J] init 0;
 C_leave: [0..N] init 0;
 i : [1..k+1];

 [join]   C_join<J-1 -> (C_join'=C_join+1);
 [joinR]  C_join=J-1-> (C_join'=0) & (C_leave'=0) & (i'=1) & (Comp'=false);
 [leave]  C_leave<N-1 -> (C_leave'=C_leave+1);
 [leaveC] C_leave<N-1 -> (C_leave'=C_leave+1) & (Comp'=true);
 [leaveR] C_leave=N-1-> (C_join'=0) & (C_leave'=0) & (i'=1) & (Comp'=false);
 [message] true  -> true;
 [messageC] true -> (Comp'=true);
 []   i < k -> k/mean :   (i'=i+1);
 [reset] i = k -> k/mean : (C_join'=0)&(C_leave'=0)&(i'=1)&(Comp'=false);
endmodule
```

C.2.8 Hybrid Key Update excluding TB- *model-LM-hybrid/tb.sm*

```
ctmc

const int J; // threshold for number of joins
const int N; // threshold for number of leaves
const int MSG; // threshold for number of messages

// values for ha
const int Max=20;
const double R_leave  = 1/365; // average leave: 1 per year
const double R_join   = 1/7;   // average join: 1 every week
const double P_comp   = 1/100; // risk of key leakage
const double R_message  = 1/30; // average communication

module DEVICES
 Size: [0..Max] init Max;

 [join]   Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [joinR]  Size<Max  -> R_join*(Max-Size):       (Size'=Size+1);
 [leave]  Size>0    -> R_leave*(1-P_comp)*Size: (Size'=Size-1);
 [leaveC] Size>0    -> R_leave*P_comp*Size:     (Size'=Size-1);
 [leaveR] Size>0    -> R_leave*Size:      (Size'=Size-1);
 [message] Size>0   -> R_message*(1-P_comp)*Size:      true;
 [messageC] Size>0  -> R_message*P_comp*Size:      true;
 [messageR] Size>0    -> R_message*Size:      true;
endmodule

module COORDINATOR
 Comp: bool init false;
 C_join: [0..J] init 0;
 C_leave: [0..N] init 0;
 C_msg: [0..MSG] init 0;

 [join]   C_join<J-1 -> (C_join'=C_join+1);
 [joinR]  C_join=J-1-> (C_join'=0)&(C_leave'=0)&(C_msg'=0)&(Comp'=false);
 [leave]  C_leave<N-1 -> (C_leave'=C_leave+1);
 [leaveC] C_leave<N-1 -> (C_leave'=C_leave+1) & (Comp'=true);
 [leaveR] C_leave=N-1-> (C_join'=0)&(C_leave'=0)&(C_msg'=0)&(Comp'=false);
 [message] C_msg<MSG-1  -> (C_msg'=C_msg+1);
 [messageC] C_msg<MSG-1 -> (C_msg'=C_msg+1) & (Comp'=true);
 [messageR] C_msg=MSG-1 -> (C_join'=0)&(C_leave'=0)&(C_msg'=0)&(Comp'=false);
endmodule
```

C.3 Reward Structures

All rewards structures actually end with `endrewards`, however they are omitted below for saving space.

```
rewards "time"
  true : 1;

rewards "Compromise FOR C.1.*"
  [leaveC] !Comp:  1;

rewards "Replacements FOR C.*.1"
  [joinR] true:  1;

rewards "Replacements FOR C.*.2"
  [leaveR] true:  1;
  [joinR] true:  1;

rewards "Replacements FOR C.*.3"
  [leaveR] true:  1;

rewards "Replacements FOR C.*.4"
  [messageR] true:  1;

rewards "Replacements FOR C.*.5"
  [reset] true:  1;

rewards "Leaves FOR C.*.1, C.*.4,
   C.*.5, C.2.6, C.2.7, C.2.8"
  [leave]  true: 1;
  [leaveC] true: 1;

rewards "Useful_Resets FOR C.*.1"
  [joinR] Comp:  1;

rewards "Useful_Resets FOR C.*.2"
  [leaveR] Comp:  1;
  [joinR] Comp:  1;

rewards "Useful_Resets FOR C.*.3"
  [leaveR] Comp:  1;

rewards "Useful_Resets FOR C.*.4"
  [messageR] Comp:  1;

rewards "Useful_Resets FOR C.*.5"
  [reset] Comp:  1;

rewards "Useless_Resets FOR C.*.1"
  [joinR] !Comp:  1;

rewards "Useless_Resets FOR C.*.2"
  [leaveR] !Comp:  1;
  [joinR] !Comp:  1;

rewards "Useless_Resets FOR C.*.3"
  [leaveR] !Comp:  1;

rewards "Useless_Resets FOR C.*.4"
  [messageR] !Comp:  1;

rewards "Useless_Resets FOR C.*.5"
  [reset] !Comp:  1;

rewards "Recovery"
  Comp:  1;

rewards "Compromise FOR C.2.*"
  [leaveC] !Comp:  1;
  [messageC] !Comp:  1;

rewards "Replacements FOR C.2.6"
  [joinR] true:  1;
  [leaveR] true:  1;
  [messageR] true:  1;
  [reset] true:  1;

rewards "Replacements FOR C.2.7"
  [joinR] true:  1;
  [leaveR] true:  1;
  [reset] true:  1;

rewards "Replacements FOR C.2.8"
  [joinR] true:  1;
  [leaveR] true:  1;
  [messageR] true:  1;

rewards "Leaves FOR C.*.2, C.*.3"
  [leave]  true: 1;
  [leaveC] true: 1;
  [leaveR] true: 1;

rewards "Useful_Resets FOR C.2.6"
  [joinR] Comp:  1;
  [leaveR] Comp:  1;
  [messageR] Comp:  1;
  [reset] Comp:  1;

rewards "Useful_Resets FOR C.2.7"
  [joinR] Comp:  1;
  [leaveR] Comp:  1;
  [reset] Comp:  1;

rewards "Useful_Resets FOR C.2.8"
  [joinR] Comp:  1;
  [leaveR] Comp:  1;
  [messageR] Comp:  1;

rewards "Useless_Resets FOR C.2.6"
  [joinR] !Comp:  1;
  [leaveR] !Comp:  1;
  [messageR] !Comp:  1;
  [reset] !Comp:  1;

rewards "Useless_Resets FOR C.2.7"
  [joinR] !Comp:  1;
  [leaveR] !Comp:  1;
  [reset] !Comp:  1;

rewards "Useless_Resets FOR C.2.8"
  [joinR] !Comp:  1;
  [leaveR] !Comp:  1;
  [messageR] !Comp:  1;
```

C.4 Stochastic Temporal Logic Formulae

```
const double T;

P=? [ F[30*T,30*T] Comp ]

S=? [ Comp ]

P=? [ Comp U>=(30*T) !Comp {Comp}{max} ]

R{"Useful_Resets"}=? [ S ]

R{"Useless_Resets"}=? [ S ]

R{"Replacements"}=? [ C<=30*T ]

R{"Leaves"}=? [ C<=30*T ]

P=? [ F[T,T] Comp ]

R{"Recovery"}=? [ C<=30*T ]

R{"Compromise"}=? [ C<=30*T ]

R{"Recovery"}=? [ C<=30*T ]/R{"Compromise"}=? [ C<=30*T ]

R{"Recovery"}=? [I=30*T]

(100*R{"Useful_Resets"}=? [S])/(R{"All_Resets"}=? [S])

(100*R{"Useless_Resets"}=? [S])/(R{"All_Resets"}=? [S])
```

Appendix D

Key Update Strategies

D.1 Characteristics of Key Update Strategies

In this appendix, we will discuss the characteristics of the key update strategies that we have been working on. We will describe some guidelines on when to choose which strategy. Indisputably, it is hard to come up with a direct solution for a scenario but we will at least explain the strengths and the weaknesses of the key update strategies we considered. We presented the pros and cons of the key update strategies in two different perspectives: *networking* and *verification*. We arranged this discussion in a separate table for each update strategy as follows: JB in Table D.1, LB in Table D.2, JLB in Table D.3, MB in Table D.4, TB in Table D.5, and Hy in Table D.6.

Under each table, we added two figures that would illustrate the general behaviour of the key update strategy on transient probability of the network key being compromised. The figure above always shows the characteristics when LM (key compromise by both device leave and communication) models are used, and the figure above always shows the characteristics when L (key compromise with leaving devices only) models are used. We configured all the model checkings with the same common parameters, i.e. `R_join`=1/180, `R_leave`=1/180, `R_message`=1/15, `P_comp`=1/1000, and `Size`=10. The threshold values vary in order to get more different results, and are available on the figures.

Table D.1: Join-based Key Update - *Strength & Weakness*

	Networking Perspective	**Verification Perspective**
Strong	In a network where number of join events is reasonably high, JB will perform better and have more control on the key compromise probability adjustment. Minimum threshold values can offer very high security, provided that compromises mostly happen by joining and leaving devices.	JB is very compact in model size, therefore one of the strongest strategies against state-space explosion.
Weak	Naturally, a join event can only happen whenever there is room for a new device. If a network is fully utilized, then a join event is strictly bound to a leave event. Therefore, in such cases not only the rate of join but also rate of leave should be considered when assigning thresholds.	JB does not exhibit a significant weakness in terms of verification.

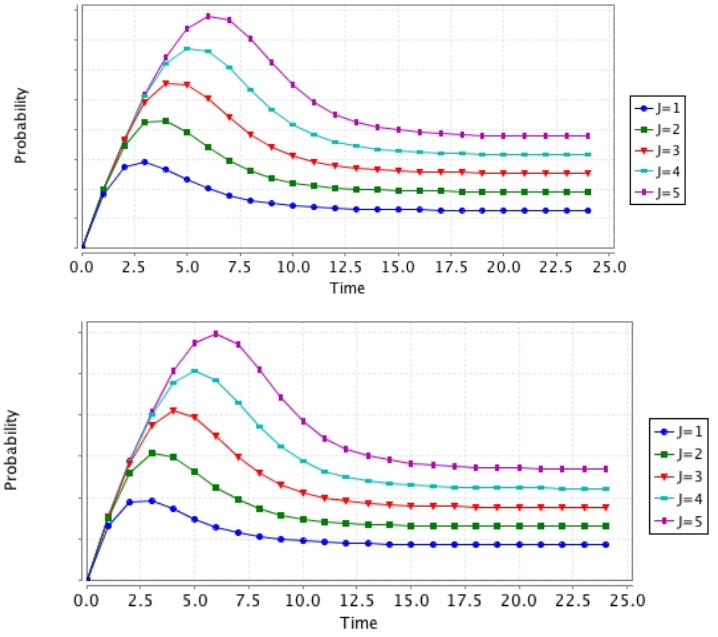


Figure D.1: Join-based Key Update - *Characteristics*

Table D.2: Leave-based Key Update - *Strength & Weakness*

	Networking Perspective	**Verification Perspective**
Strong	LB was inspired by the leaving devices who still possessed the valid network key. Therefore is very useful when a network is not stationary but dynamic.	LB is a very compact model, like JB; therefore can successfully be used in verifying very large networks.
Weak	Whenever a network is stationary, LB will fire less key updates which might be insufficient for high security concerns.	LB does not exhibit a significant weakness in terms of verification.

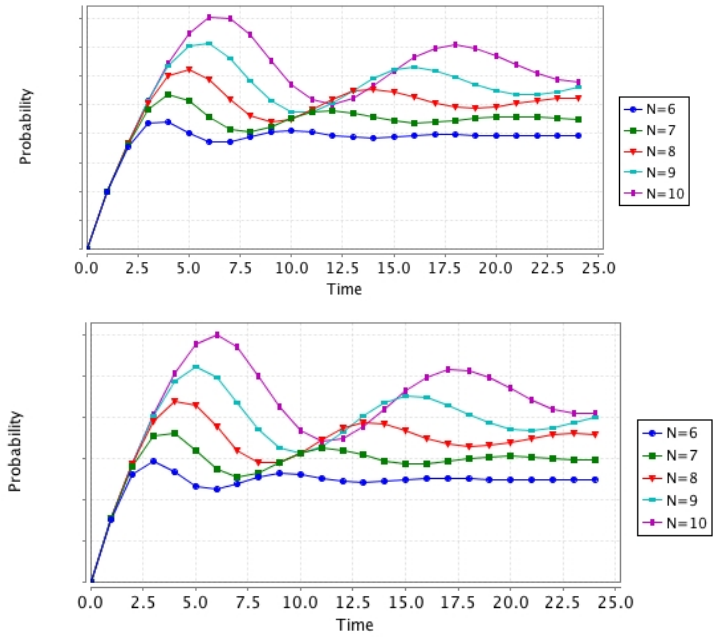


Figure D.2: Leave-based Key Update - *Characteristics*

Table D.3: Join-Leave-based Key Update - *Strength & Weakness*

	Networking Perspective	**Verification Perspective**
Strong	JLB is powered by both JB and LB key update strategies therefore employs the strong parts of these two. Since both join and leave events are considered, a threshold value is twice more sensitive than JB or LB. In other words, we have more precision on the key update thresholds.	JLB is larger than JB and LB as a model but still produces less state space than TB, MB, and Hy key update strategies.
Weak	If the majority of the key compromise events occurs from the communication over the network, then JLB might not be the optimum strategy or the threshold should be minimized to deal with this problem.	JLB is not as compact as the models that inspired it, namely JB and LB.

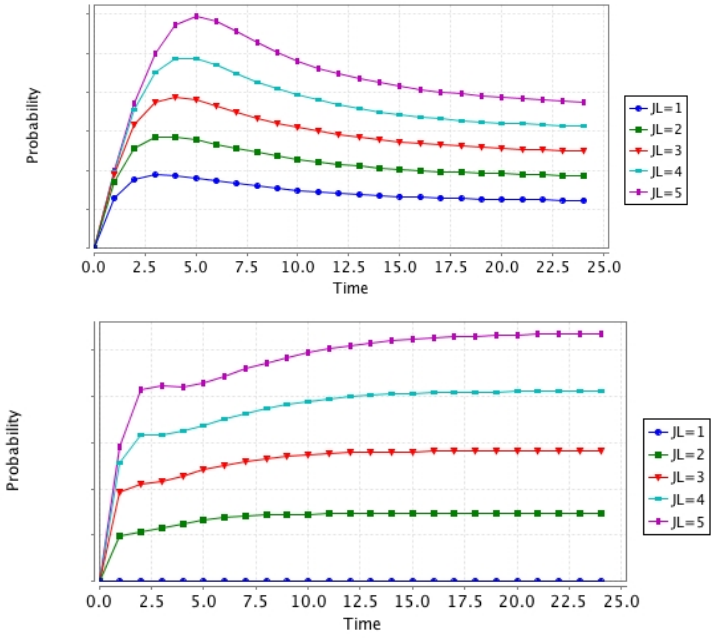


Figure D.3: Join-Leave-based Key Update - *Characteristics*

Table D.4: Message-based Key Update - *Strength & Weakness*

	Networking Perspective	**Verification Perspective**
Strong	MB is very strong when majority of the key compromise events are caused by the messages over the network.	MB is only better than Hy, in terms of state-space and model checking time.
Weak	MB does not consider leaving (and joining) devices, therefore is not aware of a device left with a valid key. Thus, in dynamic environments MB itself is not strong.	Number of messages that are communicated in a network for a certain time amount is directly proportional to the network size and messaging activity per device, therefore can quickly reach to very large numbers. This causes a big burden on the stochastic model checking since large numerical threshold values quickly exceed time and memory limits. We easily observed situations like state-spaces of 10^8, required memory of 3GB and more than an hour for a simple model checking.

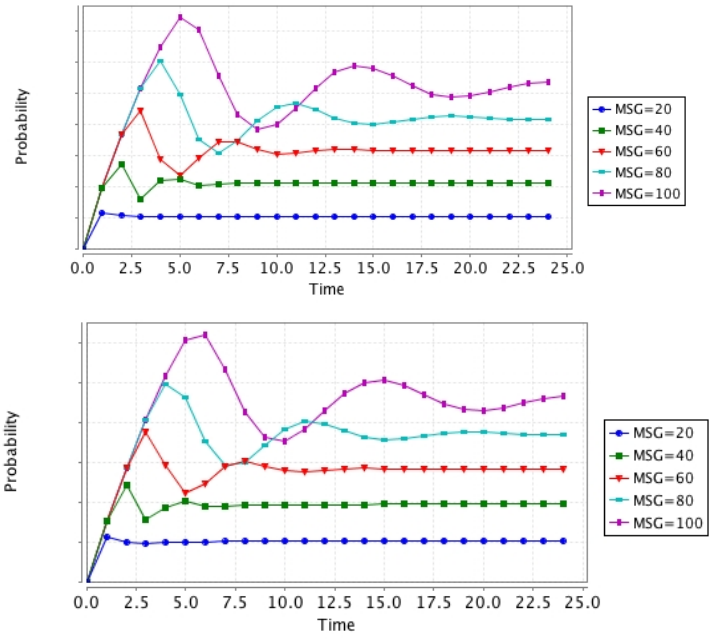


Figure D.4: Message-based Key Update - *Characteristics*

Table D.5: Time-based Key Update - *Strength & Weakness*

	Networking Perspective	**Verification Perspective**
Strong	This classical key update strategy is strong because it is periodical. Namely, it is easy to foresee how many key updates will take place, and it is easy to guess the impact of a change on the threshold value.	If implemented with exponential probability distribution then it is a very compact model. However, this is not a real strength since exponential distribution is not good at deterministic delays e.g. periodical updates. Still, it can be useful for steady-state probability computations.
Weak	TB does not consider any source of key compromise, its mere concern is the time passed. Therefore cannot easily be fitted to a specific network that has properties related to number of messages, number of joining/leaving devices etc.	We implement the TB model with phase-type distribution therefore the model size has grown at least by factor of 100 (depending on the shape parameter, which we decided to take either 100 or 1000). This makes TB not very feasible for verifying large networks. However, produced state-space and model checking times are still better than MB.

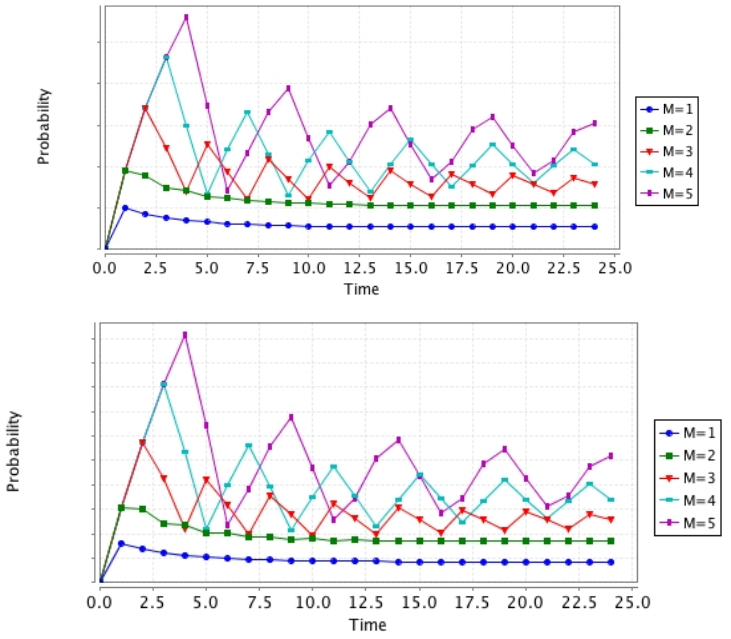


Figure D.5: Time-based Key Update - *Characteristics*

Table D.6: Hybrid Key Update - *Strength & Weakness*

	Networking Perspective	**Verification Perspective**
Strong	This strategy incorporates the strengths of all the other strategies excluding JLB, therefore we consider it to be the strongest one. An update is issued by considering the time passed, the number of devices joined, the number of devices left, and the number of messages sent.	Hy is very modular, it is easy to add a new key update strategy, or remove an unwanted key update strategy.
Weak	Implementing this strategy on real devices require much more resource than implementing only one of the strategies above. Therefore, if the coordinator device (that is running the trust center application) that is responsible of key updates is not strong enough, then Hy won't be feasible to choose.	Technically, the maximum state-space is produced in Hy models, and in parallel with that the time needed to complete one model checking is larger than in any other strategy. However, we can exclude some of the heavy strategies such as TB or MB (just like we did in the Case Study 3) to customize Hy to have more performance.

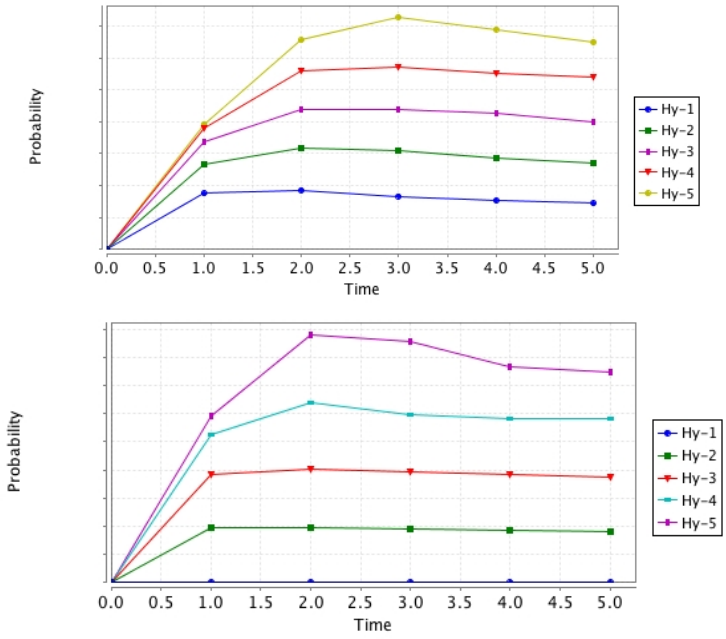


Figure D.6: Hybrid Key Update - *Characteristics*

D.2 Fluctuations in Key Compromise

In this section, we discuss the fluctuations that we observe in the nonperiodical key update methods (e.g. LB, JB). We will work on a small instantiation of JB model to find out more details about this behaviour.

We start our work with configuring the simple JB model, where the only source of key compromise is leaving devices. We will keep the state space minimal in order to visually see the transitions and manually verify the computations. As we have done previously, we will set the basic parameters that are needed to construct the state space as: `Max`=2 and `C_leave`=2. Then we will assign arbitrary values for the rates: `R_leave`=0.5, `R_join`=0.5, `P_comp`=0.01. These values are sufficient to observe fluctuations in the transient probabilities of the network key getting compromised. We present the graphical results for the transient behaviour of this model in Fig. D.7, where x-axis represent the time in *days* and we computed the transient probability every 0.25 day.

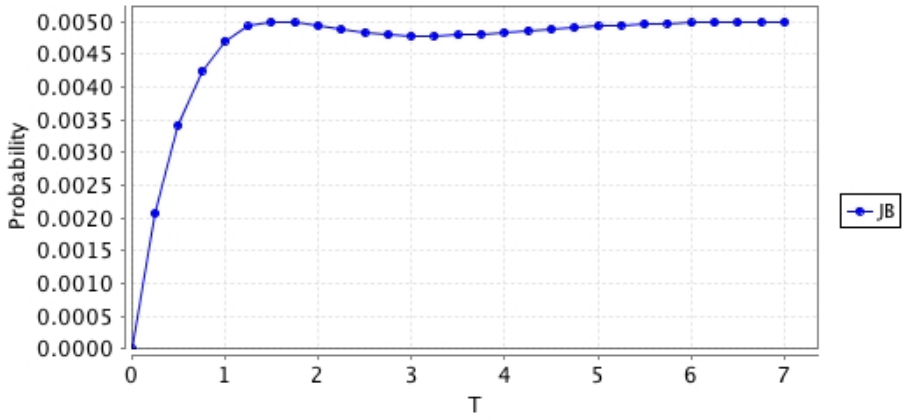


Figure D.7: Fluctuations in the transient behaviour of a simple LB model.

Even though not very significant, we can still observe the fluctuations in Fig. D.7. The probabilistic result increases after the start of the system, but then enters a decreasing period before completing the first day. After that, it starts increasing again until it reaches the steady–state. We present the precise results in Table D.7.

Now we can start to analyze the source of these transient probabilities. We first sketch the state-transition diagram of the resulting CTMC and decorate it with the transition probabilities instead of rates. We present the resulting diagram

Table D.7: Fluctuations in transient probability

Time	Prob.	Time	Prob.	Time	Prob.	Time	Prob.
0	0	0.25	0.00207	0.5	0.00342	0.75	0.00425
1	0.00471	1.25	0.00493	1.5	0.00500	1.75	0.00499
2	0.00494	2.25	0.00488	2.5	0.00484	2.75	0.00480
3	0.00479	3.25	0.00479	3.5	0.00480	3.75	0.00482
4	0.00484	4.25	0.00486	4.5	0.00489	4.75	0.00491
5	0.00493	5.25	0.00495	5.5	0.00497	5.75	0.00498
6	0.00498	6.25	0.00499	6.5	0.00499	6.75	0.00500

in Fig. D.8.

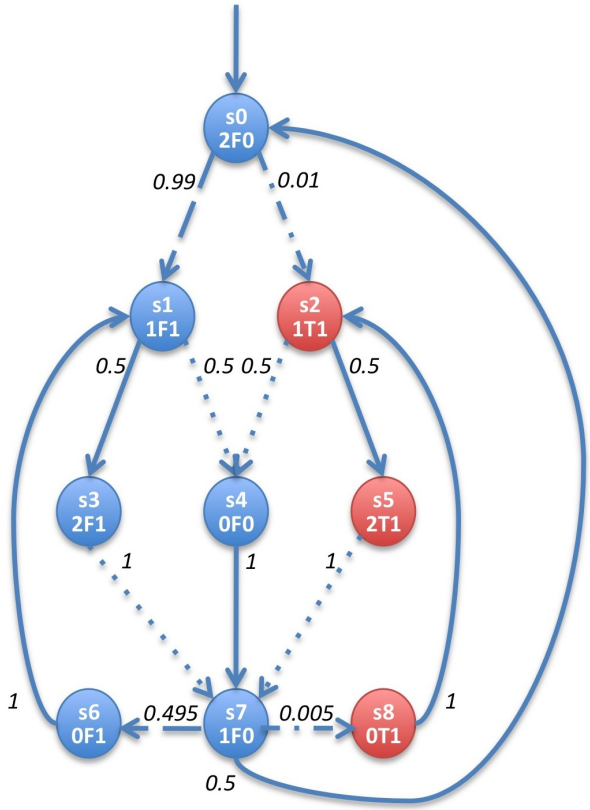


Figure D.8: Fluctuations in the transient behaviour of a simple LB model.

We have named the states as $s0 \ldots s10$ and below them we have added the

values of the variables in a concatenated format: <Size Comp C_leave>. For example the initial state is labelled as $s0$ and it is where Size=2, Comp=False, and C_leave=0.

Besides, we distinguished between the transitions as we did before, namely dashed line for leave, dashed-and-dotted line for leaveC, dotted line for leaveR, and solid line for join actions.

We computed the transition probabilities as we have explained in the preliminaries chapter, namely we divided the rate of a transition to the exit rate of the origin state. For example, the transition from state $s0$ to state $s1$ which is a leave has the probability:

$$\frac{\texttt{R\_leave}.(1-\texttt{P\_comp}).\texttt{Size}}{(\texttt{R\_leave}.(1-\texttt{P\_comp}).\texttt{Size})+(\texttt{R\_leave}.\texttt{P\_comp}.\texttt{Size})} = 1-\texttt{P\_comp}$$

which will evaluate to 0.99 if we substitute the value of P_comp we defined above.

Notice that, this example is a very special case where the transition probabilities are unintentionally equal to the transition rates. For instance, state $s0$ has two outgoing transitions which have rates 0.99 ($s0 \longrightarrow s1$, computed by R_leave.(1-P_comp).Size) and 0.01 ($s0 \longrightarrow s2$, computed by R_leave.P_comp.Size). The exit rate makes 1, thus division by exit rate result in probabilities that are equal to the rates. This situation arises from the selection of the individual rates as 0.5. If we have chosen different individual rates, for instance R_leave=0.1 (instead of 0.5), then the rates of transitions $s0 \longrightarrow s1$ and $s0 \longrightarrow s2$ would change as 0.198 and 0.002, whereas the transition probabilities would remain the same.

Now the question is what is the probability of reaching the states where the key is compromised. We have colored those states in red, in Fig. D.8.

Table D.8: Simulations

Time	State	Sojourn	Time	State	Sojourn	Time	State	Sojourn
0.0	$s0$	0.5	0.0	$s0$	0.7	0.0	$s0$	0.3
0.5	$s2$	0.5	0.7	$s1$	0.2	0.3	$s1$	0.6
0.8	$s4$	1.2	0.9	$s4$	2.2	0.9	$s4$	1.5
2.0	$s7$	1.0	3.1	$s7$	1.3	2.4	$s7$	1.8
3.0	$s0$	0.4	4.4	$s8$	0.6	4.2	$s0$	1.3
3.4	$s1$	1.2	5.0	$s2$	1.3	5.5	$s1$	0.6
4.6	$s3$	1.0	6.3	$s4$	0.7	6.1	$s4$	0.8
5.6	$s7$	0.1				6.9	$s7$	0.1
5.7	$s0$	1.3						

In PRISM we are able to simulate the model to have a sort of sanity check. We present the details of three arbitrary simulations in Table D.8. All the simulations cover the first seven days, as in Fig. D.7. We have provided the *Time* that we have reached a state, and the *Sojourn* time on that state, both in days. The red colored states are where the key is compromised for again red colored sojourn times. The green time instants are where the key is updated. In the first simulation in Table D.8, the key remained compromised for a total of 0.5 days and it was updated twice, in the first seven days. In the second simulation, the key remained compromised for a total of 1.9 days and it was updated twice, as well. In the third simulation, we don't observe any key compromise but again we have two key updates.

Then we would like to know about the probability of being in a certain state at a certain time. We can query that using PRISM (e.g. `P=? [F[T,T] (Size=2) & (!(Comp)) & (C_leave=0)]` for state $s0$), and if we repeat this for each state in the model then we can get the results in Fig. D.9. The results are in parallel with probabilities (and rates) in the model in Fig. D.8.

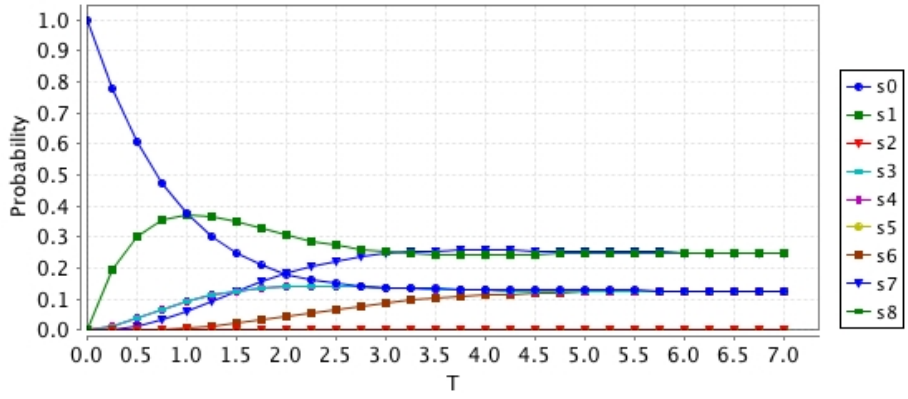


Figure D.9: Probability of being in a state at time `T`.

We can filter the results such that we query the probability of being in a compromised state at a specific time instant. As shown in Fig. D.10, the results tell us that if we are in a compromised state then it is most likely that we are in $s2$. Besides, after the initial phase the probability of being in state $s8$ increases and this causes a drop in the probability of being in $s2$.

For a deeper analysis we can compute the probability of reaching a compromised state from a safe state, at a specific time. To do this, we consider each *safe* state and find the results for reaching a compromised state such as `P=? [ F[T,T] (Comp & (C_leave=1)) {Size=1 & !Comp & C_leave=1}]` for $s1$ as

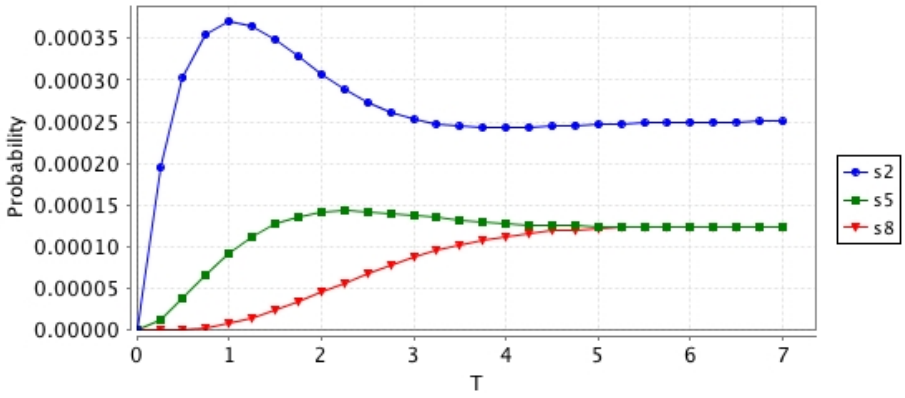


Figure D.10: Probability of being in a *compromised* state at time T.

the originating state. We present the results in Fig. D.11. Here we get interesting results, in the first part of the time period the risk in maximum if we are coming from state $s0$. and then the risk is descending as $s7$, $s3$, $s1$, and $s6$. Besides we observe fluctuations until a steady–state is reached.

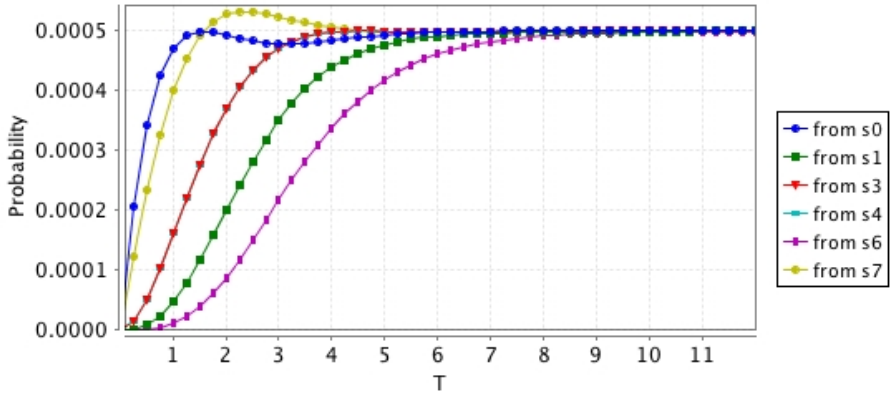


Figure D.11: Probability of reaching a compromised state from a safe state at time T.

Appendix E

Key Update Analysis in MATLAB

In this appendix, we present our MATLAB models that we used in the toolkit for LB key updates.

E.1 eexp

```
% Main function for CSL Time-bounded Until MC
% Input parameters:
% n,m,lambda,mu,gamma -> passed to eqgen
% if the rates are all 0 then predefined rates are used
% t0,t1 -> set of values to be passed to emc one by one
% varargin -> the optional parameters (SC,SCpar) passed to emc
% if no optional parameters are given: SC=1 SCpar=10^=28
% Example: eexp(20,5,0,0,0,1,10)
function eexp(n,m,lambda,mu,gamma,t0,t1,varargin)

clc;
fprintf('\n*************************************************************\n');
fprintf('* This program model checks the CSL Time-bounded Until    *\n');
fprintf('* formula for the ZigBee Leave-based Key Update Scenario  *\n');
fprintf('*                                                         *\n');
fprintf('* Ender Yuksel, DTU Informatics, ey@imm.dtu.dk, Dec. 2009 *\n');
```

```
fprintf('************************************************************\n');

% predefined values for rates and stopping criterion
if(mu==0 && lambda==0 && gamma==0)
        lambda=1/7; mu=99/36500; gamma=1/36500;
end
if(nargin==7)
        SC=1;
        SCpar=10^-28;
end
fprintf('\nMODEL DETAILS\n---------------------------------------\n');
fprintf(' Num. of devices: %d, Key update threshold: %d\n',n,m);
fprintf(' Rejoin intensity (lambda): %f\n', lambda);
fprintf(' Safe leave intensity (mu): %f\n', mu);
fprintf(' Comp. leave intensity (gamma): %f\n',gamma);

global GLOBALQ
% GLOBALQ will be computed in eqgen

tic;

global NUMOFSTATES
NUMOFSTATES=(n+1)*(2*m+1);

%% construct phi (Comp)
B=zeros(NUMOFSTATES,1);
unsafe=(m+1)*(n+1);
for i=unsafe+1:NUMOFSTATES
B(i,1)=1;
end

fprintf('\nUNIFORMISATION AND CREATION OF MATRICES\n-------------------\n');
A=eqgen(n,m,lambda,mu,gamma);
fprintf(' E,q,Q,A, and B are computed. ');
toc;

% compute the probabilities for a set of t values
fprintf('\nMODEL CHECKING\n-----------------\n');
Results=zeros(t1-t0,2);
ElapsedTime=0;
TotalTime=0;
fprintf(' Termination Criterion: ');
if(SC==0)
fprintf('Limited number of iterations, limit: %d\n',SCpar);
elseif(SC==1)
fprintf('Convergence, Epsilon: %e\n',SCpar);
end

for t=t0:t1
tic;
Temp=emc(A,B,GLOBALQ,t,SC,SCpar);
ElapsedTime=toc;
TotalTime=TotalTime+ElapsedTime;
fprintf(', Time: %fsec.\n',ElapsedTime);
Results(t,1)=t;
```

```
Results(t,2)=Temp(n+1,1);
end
fprintf('All model checkings are computed. Total time: %fsec.',TotalTime);

fprintf('\n\nRESULTS\n-----------------\n t\t probability\n');
for j=t0:t1
        fprintf(' %d \t %1.24f \n', j, Results(j,2));
end

[maxResult,indexMaxResult]=max(Results(:,2));
fprintf('\n Max: %f Index: %.1f \n',maxResult, indexMaxResult);
```

E.2 eLBtra

```
% Automated tool for computing maximum risk in LB Key Update
% Using  CSL Time-bounded Until MC
%
% Input parameters:
% n:  maximum size of the network
% m_init, m_step, m_end: set of threshold values
%      IMPORTANT: m_step CANNOT BE ZERO
%               TO CHECK SINGLE THRESHOLD WRITE A NONZERO STEP
% lambda, gamma: rates of join and cleave
% if the rates are all 0 then predefined rates are used
% t0,t1 -> set of values to be passed to emc one by one
%
% Example: elbtra(20,5,0,0,0)
function elbtra(n,m_init,m_step,m_end,lambda,gamma)

clc;
fprintf('\n*************************************************************\n');
fprintf('* eLBtra  Key Update Assistant for LB method              *\n');
fprintf('*                                                         *\n');
fprintf('* This program computes the maximum risk that a network   *\n');
fprintf('* employing the Leave-based Key Update strategy can face  *\n');
fprintf('*                                                         *\n');
fprintf('* Ender Yuksel, DTU Informatics, ey@imm.dtu.dk, Dec. 2009 *\n');
fprintf('*************************************************************\n');

% predefined values for rates
if(lambda==0 && gamma==0)
        lambda=1/7; mu=99/36500; gamma=1/36500;
else
mu=1-gamma;
end

%stopping criteria
        SC=1;     % convergence
        SCpar=10^-28;          % epsilon
```

```
global GLOBALQ
% q will be computed in eqgen

global NUMOFSTATES

% print the results
fprintf('\n\nRESULTS for a network of maximum %d devices\n--------------',n);
fprintf('\n Threshold\t Maximum Risk\t At Time Instant\n');


% SAVE ALL THE RESULTS
% Pre allocating memory is much more efficient than a growing matrix
maxThresholds=50;
maxTimeInstants=100;
Thresholds=zeros(maxThresholds, 2);
ThresholdsRowIndex=1;
Results=zeros(maxThresholds, maxTimeInstants);
ResultsRowIndex=1;

% for all threshold values
for m=m_init:m_step:m_end

Thresholds(ThresholdsRowIndex,2)=m;
ThresholdsRowIndex = ThresholdsRowIndex + 1;

NUMOFSTATES=(n+1)*(2*m+1);

%% construct phi (Comp)
B=zeros(NUMOFSTATES,1);
unsafe=(m+1)*(n+1);
for i=unsafe+1:NUMOFSTATES
B(i,1)=1;
end

% compute A and GLOBALQ
%A=eqgen(n,m,lambda,mu,gamma);
[Screenoutput, A]=evalc('eqgen(n,m,lambda,mu,gamma)');

prevResult=-1;
currResult=0;
t=1; % time instant of model checking

while prevResult < currResult
prevResult = currResult;
%Temp=emc(A,B,GLOBALQ,t,SC,SCpar);
[ScreenOutput,Temp]=evalc('emc(A,B,GLOBALQ,t,SC,SCpar)');
currResult = Temp(n+1,1);
% save all the results
Results(ResultsRowIndex,t)=currResult;
t=t+1;
end

fprintf(' %d \t\t %1.5f \t %1.1f\n', m, prevResult, t-2);
```

```
% save all the results
ResultsRowIndex = ResultsRowIndex + 1;
end

% Continue with transient analysis
reply = input('Continue with transient analysis for specific threshold?
Y/N [Y]: ', 's');
if strcmp(reply, 'Y') | strcmp(reply, 'y')
thress = input('Enterthe threshold?: ');

% check if we already computed a part of the results for that threshold
thressindex = find(Thresholds(:,2) == thress);
if ~ isempty(thressindex)
t=1;
ssepsilon=10^-4; % epsilon for reaching the steady--state
% print the results
fprintf('\n\nTransient Analysis for threshold %d (until difference %f) \n
--------------',thress, ssepsilon);
fprintf('\n Time\t Risk \n');
% print the already computed results
while Results(thressindex,t) ~= 0
        fprintf(' %d \t %1.10f \n', t, Results(thressindex,t));
t = t + 1;
end

% continue with remaining t values UNTIL WE GET EQUAL RESULTS (EPSILON)
prevResult=-1;
currResult=0;
while abs(currResult - prevResult) > ssepsilon
prevResult = currResult;
[ScreenOutput,Temp]=evalc('emc(A,B,GLOBALQ,t,SC,SCpar)');
currResult = Temp(n+1,1);
fprintf(' %d \t %1.10f \n', t, currResult);
t=t+1;
end
% user selected a different threshold
% same computation but starting from t=1
else
fprintf('\n You chose a different thresholds than above,
please restart the program');
end
end
```

E.3 eqgen

```
% This function generates the infinitesmal generator matrix
% for the specific leave based key update scenario
% therefore expecteed to be more efficient than Prism's
% INPUTS: n: number of devices
%         m: leave threshold
```

```
function A=eqgen(n,m,lambda,mu,gamma);

% compute E0
E0=zeros(n+1,1); mpg=mu+gamma;
for i=0:n E0(i+1,1)=(n-i)*lambda + i*mpg;    end

% compute E (repeat E0, vertically, that many times)
E=repmat(E0,(2*m+1),1);

% construct the submatrices
% construct Au
Au=zeros(n+1,n+1);
for j=1:n Au(j+1,j)=j;   end

% construct Alambda
Alambda=zeros(n+1,n+1);
for k=1:n
Alambda(k,k)=-E0(k,1);Alambda(k,k+1)=(n-(k-1))*lambda;
end
Alambda(n+1,n+1)=-E0(n+1,1);

% construct Amu
Amu=Au*mu;
% construct Agamma
Agamma=Au*gamma;
% construct Ampg
Ampg=Au*mpg;

% construct Q1
zero=zeros(n+1,n+1);
  % nonzero portion
  tempnon=horzcat(Alambda,Amu);
  % create the rows
  for i=1:m+1
    %% create prezeros
    if(i==2)   temppre=zero;
    elseif(i>2 && i<m+1) temppre=zero;  for j=1:i-2
    temppre=horzcat(temppre,zero); end
    elseif(i==m+1)  temppre=zero;  for j=1:m-2
    temppre=horzcat(temppre,zero); end
  end
  %% create postzeros
  if(i==m-1)   temppos=zero;
  elseif(i>0 && i<m-1)
     if(n-i==1) temppos=zero;
     else temppos=zero; for k=1:m-i-1
     temppos=horzcat(temppos,zero);  end
     end
  end
  %% concat
  if(i==1)
temprow=horzcat(tempnon,temppos);
tempcol=temprow;
  elseif(i>1 && i<m)
```

```
temprow=horzcat(temppre,tempnon,temppos);
tempcol=vertcat(tempcol,temprow);
  elseif(i==m)  temprow=horzcat(temppre,tempnon);
  tempcol=vertcat(tempcol,temprow);
  else   % i = m+1
temprow=horzcat(Ampg,temppre,Alambda);
tempcol=vertcat(tempcol,temprow);
  end
end
Q1=tempcol;

% construct Q2
Q2=zeros((m+1)*(n+1),m*(n+1));
for outerow=1:m
  for row=1:n+1
    for col=1:n+1
    Q2((outerow-1)*(n+1)+row,(outerow-1)*(n+1)+col)=Agamma(row,col);
    end
  end
end

% construct Q3
Q3=zeros(m*(n+1),(m+1)*(n+1));
for row=1:n+1
for col=1:n+1   Q3((m-1)*(n+1)+row,col)=Ampg(row,col);  end
end

% construct Q4
Q4=zeros(m*(n+1),m*(n+1));
zero4=zeros(n+1,n+1);
  % nonzero portion
  tempnon4=horzcat(Alambda,Ampg);
  % create the rows
  for i=1:m
%disp(i);
    %% create prezeros
    if(i==2)      temppre4=zero4;
    elseif(i>2 && i<m+1) temppre4=zero4;
    for j=1:i-2 temppre4=horzcat(temppre4,zero4);  end
  end
  %% create postzeros
  if(i==m-2)      temppos4=zero4;
  elseif(i>0 && i<m-2) temppos4=zero;
  for k=1:m-(i+2) temppos4=horzcat(temppos4,zero4); end
  end
  %% concat
  if(i==1)
if(m==2)  temprow4=tempnon4;   tempcol4=temprow4;
        else       temprow4=horzcat(tempnon4,temppos4);
        tempcol4=temprow4;
end
  elseif(i>1 && i<m-1)
        temprow4=horzcat(temppre4,tempnon4,temppos4);
        tempcol4=vertcat(tempcol4,temprow4);
  elseif(i==m-1)
```

```
        temprow4=horzcat(temppre4,tempnon4);
        tempcol4=vertcat(tempcol4,temprow4);
  else   % i = m
        temprow4=horzcat(temppre4,Alambda);
        tempcol4=vertcat(tempcol4,temprow4);
  end
end
Q4=tempcol4;

% concatenate Qs and construct Q
Q=horzcat(vertcat(Q1,Q3),vertcat(Q2,Q4));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% compute q for uniformisation
global GLOBALQ
GLOBALQ=1.02*max(E);
fprintf(' Uniformisation factor (q): %f\n',GLOBALQ);

% compute Punif (A)
%%find the size first
%%%% secure half
sechal=(m+1)*(n+1);
%%%% unsecure half
unsechal=m*(n+1);
A=(Q/GLOBALQ)+eye(sechal+unsechal);
```

E.4 emc

```
% Either Stopping Criterion or Limited Number of Iterations
% can be used by setting the SC switch (0 or 1)
% the last input parameter (SCpar) is:
% epsil for SC if SC=1
% lim for iterations if SC=0
function Res=emc(A,B,q,t,SC,SCpar)

global NUMOFSTATES % defined in eexp()
global LEFTTRUNC % defined in foxglynn()
global RIGHTTRUNC % defined in foxglynn()

% CONSTANTS PRECISELY AS IN PRISM DEFAULTS
TERMINATION_MAX_ITERATIONS=15000;
TERMINATION_EPSILON=10^(-6);

% constants for Fox/Glynn
underflow=1.0e-300;
overflow=1.0e+300;
accuracy=TERMINATION_EPSILON/8.0;

% compute qt
```

```
qt=q*t*30;

% create the matrices
Res=zeros(NUMOFSTATES,1);
Iter=zeros(NUMOFSTATES,1);
expqt=exp(qt);
%%% compute using fox-glynn instead
foxglynn(qt, underflow, overflow, accuracy)
fprintf('\n Fox-Glynn: qt= %.3f,
underflow= %.1e, overflow= %.1e',qt,underflow,overflow);
fprintf('\n accuracy= %.2e, LEFT-TRUNC= %d,
RIGHT-TRUNC= %d\n\n',accuracy,LEFTTRUNC-1,RIGHTTRUNC);

if(isinf(expqt)==1) fprintf(' WARNING: exp(qt) is infinity!!!'); end
Iter=(1/expqt)*B;

% MODEL CHECKING
% case j=0
%Res=Res+(eye(NUMOFSTATES)*Iter);

% TERMINATION CRITERION SELECTION
% Case 0: Limited number of iterations, specified by SCpar
if(SC==0)
for j=1:SCpar
       Iter=(qt/j)*A*Iter;
       Res=Res+Iter;
if j>TERMINATION_MAX_ITERATIONS, break, end
end
% Case 1: Stopping Criterion
elseif(SC==1)
j=1;
AjB=B; % The last 2 parts of the inf sum when j is 0
while(j<LEFTTRUNC-1 && j<TERMINATION_MAX_ITERATIONS)
AjB=A*AjB;
j=j+1;
end

fprintf(' T=%d, Iterations part1:%d ',t,j);
while(j<=RIGHTTRUNC && j<TERMINATION_MAX_ITERATIONS)
AjB=A*AjB;
Res=Res+(poisspdf(j,qt)*AjB);
j=j+1;
end
fprintf('total:%d',j);
% improper SC switch
else
fprintf('Improper SC switch!');
end
```

E.5 foxglynn

```
% Compute poisson probabilities for uniformisation (Fox-Glynn method)
% Based on the PRISM implementation by Joachim Meyer-Kayser

function foxglynn(q_tmax, underflow, overflow, accuracy)

global LEFTTRUNC
global RIGHTTRUNC

m = uint32(floor(q_tmax));
m2 = m;
if (q_tmax == 0.0)
fprintf('Overflow: TA parameter qtmax = time * maxExitRate = 0.');
end
if (q_tmax < 25.0)  fgw_left = 0; end
if (q_tmax < 400.0) % Find right using Corollary 1 with q_tmax=400
sqrt2  = sqrt(2.0);
sqrtl  = 20;
a = 1.0025 * exp (0.0625) * sqrt2;
b = 1.0025 * exp (0.125/400);  %exp (0.0003125)
startk = 1.0/(2.0 * sqrt2 * 400);
stopk = sqrtl/(2*sqrt2);
k=startk;
while(k<=stopk)
d = 1.0/(1 - exp ((-2.0/9.0)*(k*sqrt2*sqrtl + 1.5)));
f = a * d * exp (-0.5*k*k) / (k * sqrt (2.0 * 3.1415926));
if f <= accuracy/2.0, break, end
k=k+3.0; end
if (k > stopk)  k = stopk; end
fgw_right = uint32(ceil(m2 + k*sqrt2*sqrtl + 1.5));
end
if (q_tmax >= 400.0)
% Find right using Corollary 1 using actual q_tmax
sqrt2  = sqrt (2.0);
sqrtl  = sqrt (q_tmax);
a = (1.0 + 1.0/q_tmax) * exp (0.0625) * sqrt2;
b = (1.0 + 1.0/q_tmax) * exp (0.125/q_tmax);
startk = 1.0/(2.0 * sqrt2 * q_tmax);
stopk = sqrtl/(2*sqrt2);
        k=startk;
        while(k<=stopk)
d = 1.0/(1 - exp ((-2.0/9.0)*(k*sqrt2*sqrtl + 1.5)));
f = a * d * exp (-0.5*k*k) / (k * sqrt (2.0 * 3.1415926));
if f <= accuracy/2.0, break, end
k=k+3.0; end
if (k > stopk)  k = stopk; end
fgw_right = uint32(ceil(m2 + k*sqrt2*sqrtl + 1.5));
end
if (q_tmax >= 25.0)
% Find left using Corollary 2 using actual q_tmax
sqrt2  = sqrt (2.0);
sqrtl  = sqrt (q_tmax);
a = (1.0 + 1.0/q_tmax) * exp (0.0625) * sqrt2;
b = (1.0 + 1.0/q_tmax) * exp (0.125/q_tmax);
startk = 1.0/(sqrt2*sqrtl);
```

```
stopk =  (m2 - 1.5)/(sqrt2*sqrtl);
        k=startk;
        while(k<=stopk)
if (b * exp(-0.5*k*k)/(k * sqrt (2.0 * 3.1415926))
<= accuracy/2.0),break,end
k=k+3.0; end
if (k > stopk)  k = stopk; end
fgw_left = uint32(floor(m2 - k*sqrtl - 1.5));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (fgw_left < 0)  fgw_left = 0; end
q = overflow / ((10.0^10.0) * (fgw_right - fgw_left));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LEFTTRUNC=fgw_left;
RIGHTTRUNC=fgw_right;
fgw_weights=zeros(fgw_right-fgw_left+1,1);
fgw_weights(m-fgw_left+1,1) = q;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% down
j=m;
while(j>fgw_left)
fgw_weights(j-fgw_left,1) = (j/q_tmax) * fgw_weights(j-fgw_left+1,1);
j=j-1; end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%up
if (q_tmax < 400)
if (fgw_right > 600)  fprintf('Overflow: right truncation point > 600.');
end
j=m;
while(j<fgw_right)
q = q_tmax / (j+1);
if (fgw_weights(j-fgw_left+1,1) > underflow/q)
fgw_weights(j-fgw_left,1) = q * fgw_weights(j-fgw_left+1,1);
j=j+1;
else  fgw_right = j;
end
end
else
j=m;
while(j<fgw_right)
fgw_weights(j-fgw_left,1) = (q_tmax/(j+1)) * fgw_weights(j-fgw_left+1,1);
j=j+1; end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
l = fgw_left;
r = fgw_right;
fgw_total_weight = 0.0;
while (l < r)
if (fgw_weights(l-fgw_left+1,1) <= fgw_weights(r-fgw_left+1,1))
fgw_total_weight = fgw_total_weight + fgw_weights(l-fgw_left+1,1);
l=l+1;
else
fgw_total_weight = fgw_total_weight + fgw_weights(r-fgw_left+1,1);
r=r-1; end
end
```

```
fgw_total_weight = fgw_total_weight + fgw_weights(l-fgw_left+1,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (fgw_right < 0)
fprintf('Overflow in Fox-Glynn computation (time bound too big?)');
end
for i = fgw_left:fgw_right
fgw_weights(i-fgw_left+1,1) = fgw_weights(i-fgw_left+1,1) / fgw_total_weight;
end
```

Bibliography

[Aba99] M. Abadi, "Secrecy by typing in security protocols," *J. ACM*, vol. 46, pp. 749–786, September 1999. [Online]. Available: http://doi.acm.org/10.1145/324133.324266

[AC08] A. Armando and L. Compagna, "Sat-based model-checking for security protocols analysis," *Int. J. Inf. Secur.*, vol. 7, pp. 3–32, January 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1341131.1341135

[AG97] M. Abadi and A. D. Gordon, "A calculus for cryptographic protocols: The spi calculus," *Fourth ACM Conference on Computer and Communications Security*, pp. 36–47, 1997.

[AH96] R. Alur and T. A. Henzinger, "Reactive modules," in *Formal Methods in System Design*. IEEE Computer Society Press, 1996, pp. 207–218.

[AK96] R. Anderson and M. Kuhn, "Tamper resistance: a cautionary note," in *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*. Berkeley, CA, USA: USENIX Association, 1996, pp. 1–1. [Online]. Available: http://portal.acm.org/citation.cfm?id=1267167.1267168

[Ame01] *Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American National Standards Institute Std. ANSI X9.63:2001, 2001.

[AN94] M. Abadi and R. Needham, "Prudent engineering practice for cryptographic protocols," *Technical report, SRC DIGITAL*, June 1994.

[ASDO10] S. T. Ali, V. Sivaraman, A. Dhamdhere, and D. Ostry, "Secure key loss recovery for network broadcast in single-hop wireless sensor networks," *Ad Hoc Netw.*, vol. 8, pp. 668–679, August 2010. [Online]. Available: http://dx.doi.org/10.1016/j.adhoc.2010.01.003

[ASSB96] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton, "Verifying continuous time markov chains," in *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 1996, pp. 269–276.

[BAN90] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, pp. 18–36, 1990.

[BB02] M. Boreale and M. G. Buscemi, "Experimenting with sta, a tool for automatic analysis of security protocols," in *Proceedings of the 2002 ACM symposium on Applied computing*, ser. SAC '02. New York, NY, USA: ACM, 2002, pp. 281–285. [Online]. Available: http://doi.acm.org/10.1145/508791.508847

[BBD+03] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson, "Automatic valication of protocol narration," *Procceeding of Computer Security Foundations Workshop XVI*, pp. 126–140, 2003.

[BBD+05] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson, "Static validation of security protocols," *Journal of Computer Security*, vol. 13(3), pp. 347–390, 2005.

[BCP09] M. Benerecetti, N. Cuomo, and A. Peron, "Tpmc: A model checker for time-sensitive security protocols," *JCP*, vol. 4, no. 5, pp. 366–377, 2009.

[BDM98] P. Behm, P. Desforges, and J.-M. Meynadier, "MÉtÉor: An industrial success in formal development," in *B '98: Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method*. London, UK: Springer-Verlag, 1998, p. 26.

[BE69] V. A. Busam and D. E. Englund, "Optimization of expressions in fortran," *Commun. ACM*, vol. 12, no. 12, pp. 666–674, 1969.

[Bel07] G. Bella, *Formal Correctness of Security Protocols (Information Security and Cryptography)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[BHHK00] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "Model checking continuous-time markov chains by transient analysis," in *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 2000, pp. 358–372.

[BHHK03] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time markov chains," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 524–541, 2003.

[BHRM09] S. Biswas, M. M. Haque, S. Rashwand, and J. Misic, "Fast, seamless rekeying in wireless sensor networks," in *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, ser. ICDCSW '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 166–171. [Online]. Available: http://dx.doi.org/10.1109/ICDCSW.2009.78

[BK08] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[BKH99] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time markov chains," in *Proceedings of the 10th International Conference on Concurrency Theory*, ser. CONCUR '99. London, UK: Springer-Verlag, 1999, pp. 146–161. [Online]. Available: http://portal.acm.org/citation.cfm?id=646734.701464

[BKHW05] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf, "Comparative branching-time semantics for markov chains," *Inf. Comput.*, vol. 200, no. 2, pp. 149–214, 2005.

[BKPA09] S. Basagiannis, P. Katsaros, A. Pombortsis, and N. Alexiou, "Probabilistic model checking for the quantification of DoS security threats," *Computers & Security*, 2009.

[Bla01] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, ser. CSFW '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 82–. [Online]. Available: http://portal.acm.org/citation.cfm?id=872752.873511

[Bla08] B. Blanchet, "A computationally sound mechanized prover for security protocols," *IEEE Trans. Dependable Secur. Comput.*, vol. 5, pp. 193–207, October 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1477074.1477550

[BMP03] G. Bella, F. Massacci, and L. C. Paulson, "Verifying the set registration protocols," *IEEE Journal of Selected Areas In Communications*, vol. 21, no. 1, pp. 77–87, 2003.

[BMV05] D. A. Basin, S. Mödersheim, and L. Viganò, "Ofmc: A symbolic model checker for security protocols," *Int. J. Inf. Sec.*, vol. 4, no. 3, pp. 181–208, 2005.

[BNP02] M. Boreale, R. D. Nicola, and R. Pugliese, "Proof techniques for cryptographic processes," *SIAM J. Comput.*, vol. 31, pp. 947–986, March 2002. [Online]. Available: http://portal.acm.org/citation.cfm?id=586840.586871

[BRNN04] M. Buchholtz, H. Riis Nielson, and F. Nielson, "A calculus for control flow analysis of security protocols," *Int. J. Inf. Secur.*, vol. 2, no. 3, pp. 145–167, 2004.

[CAS08] A. Cardenas, S. Amin, and S. S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *First International Workshop on Cyber-Physical Systems (WCPS2008)*. IEEE, June 2008. [Online]. Available: http://www.truststc.org/pubs/345.html

[CGPM05] H. Chan, V. D. Gligor, A. Perrig, and G. Muralidharan, "On the distribution and revocation of cryptographic keys in sensor networks," *IEEE Trans. Dependable Secur. Comput.*, vol. 2, pp. 233–247, July 2005. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2005.37

[CGR92] D. Craigen, S. L. Gerhart, and T. Ralston, "An international survey of industrial applications of formal methods," in *Proceedings of the Z User Workshop*. London, UK: Springer-Verlag, 1992, pp. 1–5. [Online]. Available: http://portal.acm.org/citation.cfm?id=647280.722639

[CJ97] J. A. Clark and J. L. Jacob, "A survey of authentication protocol literature," *Technical Report 1.0*, 1997.

[CJM00] E. M. Clarke, S. Jha, and W. Marrero, "Verifying security protocols with brutus," *ACM Trans. Softw. Eng. Methodol.*, vol. 9, pp. 443–487, October 2000. [Online]. Available: http://doi.acm.org/10.1145/363516.363528

[CKKP05] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan, "Model checking markov reward models with impulse rewards," in *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 722–731.

[Com91] *Information Technology Security Evaluation Criteria (ITSEC): Preliminary Harmonised Criteria*, Commission of the European Communities Std. Document COM(90) 314, 1991.

[Cra08] R. Cragie, "Former chairman of the zigbee security working group," personal communication, October 2008.

[DG04] G. Delzanno and P. Ganty, "Automatic verification of time sensitive cryptographic protocols." in *TACAS*, ser. Lecture Notes in Computer Science, K. Jensen and A. Podelski, Eds., vol. 2988. Springer, 2004, pp. 342–356.

[DHHM05] J. Deng, C. Hartung, R. Han, and S. Mishra, "A practical study of transitory master key establishment for wireless sensor networks," in *In 1st IEEE/CreateNet Conference on Security and Privacy in Communication Networks (SecureComm 2005*, 2005, pp. 289–299.

[DY81] D. Dolev and A. C. Yao, "On the security of public key protocols," in *SFCS '81: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science.* Washington, DC, USA: IEEE Computer Society, 1981, pp. 350–357.

[Fed01] *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Std. FIPS Pub 197, 2001.

[Fed02] *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standards Std. FIPS Pub 198, 2002.

[FG88] B. L. Fox and P. W. Glynn, "Computing poisson probabilities," *Commun. ACM*, vol. 31, pp. 440–445, April 1988. [Online]. Available: http://doi.acm.org/10.1145/42404.42409

[GL10] S. Guo and V. C. M. Leung, "A compromise-resilient group rekeying scheme for hierarchical wireless sensor networks," in *WCNC*, 2010, pp. 1–6.

[HKMKS00] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "A markov chain model checker," in *TACAS*, 2000, pp. 347–362.

[IEE03] *Wireless Medium Access Control and Physical Layer Specifications for Low Rate Wireless Personal Area Networks*, IEEE Std. IEEE Std. 802.15.4-2003, 2003.

[Ker83] A. Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, vol. IX, pp. 5–83, January 1883.

[KKLW07] J. P. Katoen, D. Klink, M. Leucker, and V. Wolf, "Three-valued abstraction for continuous-time markov chains," in *CAV'07: Proceedings of the 19th international conference on Computer aided verification*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 311–324.

[KKNP01] J.-P. Katoen, M. Z. Kwiatkowska, G. Norman, and D. Parker, "Faster and symbolic ctmc model checking," in *Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, ser. PAPM-PROBMIV '01. London, UK: Springer-Verlag, 2001, pp. 23–38. [Online]. Available: http://portal.acm.org/citation.cfm?id=645776.668418

[KKZ05] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker," in *Quantitative Evaluation of Systems (QEST)*. Los Alamos, CA, USA: IEEE Computer Society, 2005, pp. 243–244.

[KKZJ07] J.-P. Katoen, T. Kemna, I. Zapreev, and D. N. Jansen, "Bisimulation minimisation mostly speeds up probabilistic model checking," in *TACAS'07: Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 87–101.

[KNP04] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 6, no. 2, pp. 128–142, 2004.

[KNP07] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for the Design of Computer, Communication, and Software Systems*, ser. Lecture Notes in Computer Science, M. Bernardo and J. Hillston, Eds., vol. 4486. Springer, 2007, pp. 220–270.

[KS60] J. Kemeny and J. Snell, *Finite Markov Chains*. Springer Verlag, 1960.

[LM69] E. S. Lowry and C. W. Medlock, "Object code optimization," *Commun. ACM*, vol. 12, no. 1, pp. 13–22, 1969.

[LMPG07] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "Minisec: a secure sensor network communication architecture," in *Proceedings of the 6th international conference on Information processing in sensor networks*, ser. IPSN '07. New York,

NY, USA: ACM, 2007, pp. 479–488. [Online]. Available: http://doi.acm.org/10.1145/1236360.1236421

[Low95] G. Lowe, "An attack on the needham-schroeder public-key authentication protocol," *Inf. Process. Lett.*, vol. 56, no. 3, pp. 131–133, 1995.

[Low96] G. Lowe, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *TACAs '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*. London, UK: Springer-Verlag, 1996, pp. 147–166.

[Low97] G. Lowe, "Casper: A compiler for the analysis of security protocols," *Proceedings of the 10th Computer Security Foundations Workshop*, 1997.

[LS91] K. G. Larsen and A. Skou, "Bisimulation through probabilistic testing," *Inf. Comput.*, vol. 94, no. 1, pp. 1–28, 1991.

[LyS] The lysa website. [Online]. Available: http://www.imm.dtu.dk/English/Research/Language-Based_Technology/Research/LySa.aspx

[MAK08] J. V. Misic, F. Amini, and M. Khan, "Performance implications of periodic key exchanges and packet integrity overhead in an 802.15.4 beacon enabled cluster," *Int'l Journal of Sensor Networks*, vol. 3, no. 1, pp. 33–42, 2008.

[Mat09] (2009) Matlab version 7.3. the mathworks inc. [Online]. Available: http://www.mathworks.com

[Mea03] C. Meadows, "Formal methods for cryptographic protocol analysis: emerging issues and trends," *Selected Areas in Communications, IEEE Journal on*, vol. 21, no. 1, pp. 44 – 54, Jan. 2003.

[MFF] S. Mccanne, S. Floyd, and K. Fall, "ns2 (network simulator 2)," http://www-nrg.ee.lbl.gov/ns/. [Online]. Available: http://www-nrg.ee.lbl.gov/ns

[Mil99] R. Milner, *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.

[MMS97] J. Mitchell, M. Mitchell, and U. Stern, "Automated analysis of cryptographic protocols using mur/spl phi/," *Security and Privacy, IEEE Symposium on*, vol. 0, p. 0141, 1997.

[MP92] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems.* New York, NY, USA: Springer-Verlag New York, Inc., 1992.

[NN97] H. R. Nielson and F. Nielson, "Infinitary control flow analysis: a collecting semantics for closure analysis," *Proceeding of the 24th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pp. 332–345, 1997.

[NN98] F. Nielson and H. R. Nielson, "Flow logics and operational semantics," *Electronic Notes of Theoretical Computer Science*, vol. 10, pp. 150–169, 1998.

[NN02] H. R. Nielson and F. Nielson, "Flow logic: A multi-paradigmatic approach to static analysis. the essence of computation - complexity, analysis, transformation," *Lecture Notes in Computer Science*, pp. 223–244, 2002.

[NNH99] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis.* Springer Verlag, 1999.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, ser. LNCS. Springer, 2002, vol. 2283.

[NS78] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21(12), December 1978.

[NSF10] "Cyber-physical systems (cps) – program solicitation, nsf 10–515," 2010. [Online]. Available: http://www.nsf.gov/pubs/2010/nsf10515/nsf10515.htm

[OR87] D. Otway and O. Rees, "Efficient and timely mutual authentication," *SIGOPS Oper. Syst. Rev.*, vol. 21, pp. 8–10, January 1987. [Online]. Available: http://doi.acm.org/10.1145/24592.24594

[O'S09] N. O'Shea, "Protocol analysis in a new lyte," in *Proceedings of the 21st Nordic Workshop on Programming Theory*, M. R. Hansen and A. W. Brekling, Eds. DTU Informatics at the Techincal University of Denmark, 2009, pp. 83–94, presented at: Nordic Workshop on Programming Theory, NWPT ; 21, 2009.

[Pau98] L. C. Paulson, "The inductive approach to verifying cryptographic protocols," *J. Comput. Secur.*, vol. 6, pp. 85–128, January 1998. [Online]. Available: http://portal.acm.org/citation.cfm?id=353677.353681

[PJ10] A. R. Pais and S. Joshi, "A new probabilistic rekeying method for secure multicast groups," *Int. J. Inf. Secur.*, vol. 9, pp. 275–286, August 2010. [Online]. Available: http://dx.doi.org/10.1007/s10207-010-0108-z

[Pnu77] A. Pnueli, "The temporal logic of programs," in *SFCS '77: Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1977, pp. 46–57.

[PP07] J. I. Pagter and M. O. Pedersen, "The all-or-nothing anti-theft policy–theft protection for pervasive computing," in *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 626–631.

[Pria] The PRISM model checker website. [Online]. Available: http://www.prismmodelchecker.org/

[Prib] The PRISM manual, oxford university. [Online]. Available: http://www.prismmodelchecker.org/manual

[PST01] A. Perrig, D. Song, and J. D. Tygar, "Elk, a new protocol for efficient large-group key distribution," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 247–. [Online]. Available: http://portal.acm.org/citation.cfm?id=882495.884429

[PST+02] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "Spins: security protocols for sensor networks," *Wirel. Netw.*, vol. 8, pp. 521–534, September 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1016598314198

[QS82] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in cesar," in *Proceedings of the 5th Colloquium on International Symposium on Programming*. London, UK: Springer-Verlag, 1982, pp. 337–351.

[RLSS10] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 731–736. [Online]. Available: http://doi.acm.org.globalproxy.cvt.dk/10.1145/1837274.1837461

[SA00] F. Stajano and R. J. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks," in *Proceedings of the 7th International Workshop on Security Protocols*. London, UK: Springer-Verlag, 2000, pp. 172–194.

[Sch98] S. Schneider, "Verifying authentication protocols in csp," *IEEE Trans. Softw. Eng.*, vol. 24, pp. 741–758, September 1998. [Online]. Available: http://portal.acm.org/citation.cfm?id=287870.287882

[SS98] V. Shmatikov and U. Stern, "Efficient finite-state analysis for large security protocols," in *Proceedings of the 11th IEEE workshop on Computer Security Foundations*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 106–. [Online]. Available: http://portal.acm.org/citation.cfm?id=794198.795105

[Ste94] W. J. Stewart, *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[SVA04] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *CAV*, 2004, pp. 202–215.

[SW04] N. Sastry and D. Wagner, "Security considerations for ieee 802.15.4 networks," in *WiSe '04: Proceedings of the 3rd ACM workshop on Wireless security*. New York, NY, USA: ACM, 2004, pp. 32–42.

[Syv93] P. F. Syverson, "Adding time to a logic of authentication," in *Proceedings of the 1st ACM conference on Computer and communications security*, ser. CCS '93. New York, NY, USA: ACM, 1993, pp. 97–101. [Online]. Available: http://doi.acm.org/10.1145/168588.168600

[UBLC10] A. S. Uluagac, R. A. Beyah, Y. Li, and J. A. Copeland, "Vebek: Virtual energy-based encryption and keying for wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 9, pp. 994–1007, 2010.

[WHA99] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: Issues and architectures," United States, 1999.

[WLSC06] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless sensor network security: A survey," in *Security in Distributed, Grid, and Pervasive Computing, Yang Xiao (Eds.)*. CRC Press, 2006.

[YNN08] E. Yüksel, H. R. Nielson, and F. Nielson, "Zigbee-2007 security essentials," in *Proceedings of the 13th Nordic Workshop on Secure IT Systems*, 2008, pp. 65–82.

[YNN09a] E. Yüksel, H. R. Nielson, and F. Nielson, "Quantitative security analysis of zigbee key updates," in *Proceedings of the 21st Nordic Workshop on Programming Theory*, M. R. Hansen and A. W. Brekling, Eds. Technical University of Denmark, 2009, pp. 14–16.

[YNN09b] E. Yüksel, H. R. Nielson, and F. Nielson, "A secure key establishment protocol for zigbee wireless sensor networks," in *Proceedings of the 24th International Symposium on Computer and Information Sciences (ISCIS 2009)*. IEEE, 2009, pp. 340–345.

[YNN10a] E. Yüksel, H. R. Nielson, and F. Nielson, "A secure key establishment protocol for zigbee wireless sensor networks," *The Computer Journal*, Oxford University Press, 2010.

[YNN+10b] E. Yüksel, H. R. Nielson, F. Nielson, M. Fruth, and M. Kwiatkowska, "Optimizing zigbee security using stochastic model checking," DTU Informatics, Tech. Rep. IMM-Technical Report-2010-08, 2010.

[YNN11a] E. Yüksel, H. R. Nielson, and F. Nielson, "Characteristics of key update strategies for wireless sensor networks," in *Proceedings of the International Conference on Network Communication and Computer (ICNCC 2011)*. IEEE, 2011, p. to appear.

[YNN11b] E. Yüksel, H. R. Nielson, and F. Nielson, "Comparison of key update methods for resource-constrained networks," in *Under submission*, 2011.

[YNN+11c] E. Yüksel, H. R. Nielson, F. Nielson, M. Fruth, and M. Kwiatkowska, "Optimizing key updates in sensor networks," in *Proceedings of the IEEE Sensors Applications Symposium (SAS 2011)*. IEEE, 2011, pp. 82–87.

[You05] H. L. S. Younes, "Ymer: A statistical model checker," in *CAV*, 2005, pp. 429–433.

[Yük10] E. Yüksel. (2010) Verification tools website. [Online]. Available: http://www2.imm.dtu.dk/ey/tools

[Zig06] *ZigBee Specification*, ZigBee Alliance Std. ZigBee-2006, 053 474r13, 2006.

[Zig08a] *ZigBee Home Automation Profile Specification*, ZigBee Alliance Std. ZigBee-2007-HA, r25, 2008.

[Zig08b] *ZigBee-PRO Stack Profile Specification*, ZigBee Alliance Std. ZigBee-PRO-2007-Stack, r05, 2008.

[Zig08c] *ZigBee Smart Energy Profile Specification*, ZigBee Alliance Std. ZigBee-2007-SE, r14, 2008.

[Zig08d] *ZigBee Specification*, ZigBee Alliance Std. ZigBee-2007, 053 474r17, 2008.

[Zig08e] *ZigBee Stack Profile Specification*, ZigBee Alliance Std. ZigBee-2007-Stack, r09, 2008.

[ZLA06] J. Zheng, M. J. Lee, and M. Anshel, "Toward secure low rate wireless personal area networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 10, pp. 1361–1373, 2006.

[ZSJ03] S. Zhu, S. Setia, and S. Jajodia, "Leap: efficient security mechanisms for large-scale distributed sensor networks," in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 62–72. [Online]. Available: http://doi.acm.org/10.1145/948109.948120